НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

«На правах рукопису»
УДК 004.4

До захисту допущено:
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____20__ р.

Магістерська дисертація

на здобуття ступеня магістра
за освітньо-професійною програмою «Програмне забезпечення інформаційно-комунікаційних систем»
зі спеціальності 121 «Інженерія програмного забезпечення»
на тему: «Сервіс з агрегування платіжних систем»

Виконав (-ла):
студент (-ка) VI курсу, групи ІТ-93мп
Прилуцький Павло Валерійович _____

Керівник:
доцент кафедри АУТС, к.т.н ,
Букасов Максим Михайлович _____

Рецензент:
_____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Програмне забезпечення інформаційно-комунікаційних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Прилуцькому Павлу Валерійовичу

1. Тема дисертації «Сервіс з агрегування платіжних систем», науковий керівник дисертації Букасов Максим Михайлович, к.т.н, затверджені наказом по університету від «26» 10 2020 р. №3132-с

2. Термін подання студентом дисертації _____

3. Об’єкт дослідження: покращення взаємодії користувача із платіжними системами.

4. Вихідні дані: Технічне завдання, мова програмування PHP, СУБД – PostgreSql.

5. Перелік завдань, які потрібно розробити: Спроектувати архітектуру, описати сценарії використання системи, вибрати засоби реалізації та розробити сервіс.

6. Орієнтовний перелік ілюстративного (графічного) матеріалу: Діаграма прецендентів, структурна схема, діаграма класів, діаграма послідовності, ER-діаграма.

7. Орієнтовний перелік публікацій

8. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Видача завдання	11.10.2019	
2	Вивчення об’єкту дослідження	03.03.2020	
3	Проектування документації	04.07.2020	
4	Розробка програмної моделі	01.09.2020	
5	Реалізації сервісу	20.10.2020	
6	Оформлення документації	25.11.2020	
7	Подання роботи до попереднього захисту	02.12.2020	

Студент Павло ПРИЛУЦЬКИЙ

Науковий керівник Максим БУКАСОВ

РЕФЕРАТ

Прилуцький П.В. Сервіс з агрегування платіжних систем. КПІ ім. Ігоря Сікорського, Київ, 2020. Дисертація містить 98 сторінок, 47 рисунків, 30 таблиць, 17 літературних джерел та 8 додатків.

Актуальність: Актуальність розроблюваного сервісу полягає в наданні можливості іншим компаніям підключати та використовувати тільки одну систему для платежів, а не декілька. Оскільки компаніям затратно по часу та ресурсам підключати та підтримувати декілька платіжних провайдерів, то сервіс буде затребуваний.

Мета: Основною метою дисертації було розробити агрегатор платіжних систем, до якого можна приєднувати безліч різних платіжних провайдерів та надати інтерфейс керування ними.

Задачі: для досягнення мети були поставлені та розв’язані такі завдання:

- розробити сервіс платежів;
- розробити сервіс виплат;
- розробити маршрутизацію між платіжними системами;
- реалізувати налаштування аккаунту.

Об’єкт: об’єктом дослідження в роботі є використання онлайн платежів.

Предмет: предметом дослідження є агрегування платіжних систем.

Ключові слова: АГРЕГАТОР, ПЛАТІЖНА СИСТЕМА, ЄДИНИЙ ІНТЕРФЕЙС.

ABSTRACT

Prylutskyi P.V. Payment systems aggregation service. Igor Sikorsky Kyiv Polytechnic Institute, Kyiv, 2020. The dissertation contains 98 pages, 47 drawings, 30 tables, 17 literary sources and 8 appendixes.

Relevance: The relevance of the service is that it allows other companies to connect and use one system for payments instead of several. The service will be in demand, as companies spend a lot of time and resources connecting and supporting several payment providers.

Purpose: The main purpose of the dissertations was to build an aggregator of payment systems with the possibility to connect different payment providers into it and to provide an interface for them.

The tasks: To achieve this goal, the following tasks have been set and solved:

- develop a payment service;
- develop a payout service;
- develop routing between payment systems;
- implement account settings.

Object: the object of research is the use of online payments.

Subject matter: subject of research is aggregation of payment systems.

Keywords: AGGREGATOR, PAYMENT SYSTEM, SINGLE INTERFACE.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API – application programming interface;

GIST – generalized search tree:

HPP – hosted payment page;

JSON - javascript object notation;

ORM – object-relational mapping;

P2P – peer-to-peer;

PG – payment gateway;

POG – payout gateway;

PSP – payment service provider;

RDP – route decision point;

URL – uniform resource locator.

ВСТУП

Без грошей важко уявити сучасне суспільство. Перша згадка про гроші була ще в 3000 році до н.е., але це були не такі гроші, як в нашому теперішньому розумінні, це були хутра звірів, зброя тощо. Але однак через свою незручність у використанні найбільш вживаним еквівалентом стали такі метали, як золото та срібло, а ще згодом вони набули форму монет. Потім також був довгий шлях грошей до того вигляду, якого вони набули зараз, але не будемо на ньому зупинятися, оскільки сильних та важливих для нас змін вони не зазнали.

Наступна гілка у розвитку грошей пов’язана з появою інтернету. Оскільки стало очевидно, що якщо магазини потроху переходять до “світової павутини” то і оплачувати товари можна і без фізичного контакту з покупцем. Європейський союз визнав електронні гроші у 1994 році, а у 1996 році він почав аналізувати та контролювати їх.

Слід зазначити, що електронні гроші розділяють на анонімні та персоніфіковані. Перші це ті, які ніяк не пов’язані безпосередню з власником, тобто ніде не вказана інформація про нього, прикладом є Bitcoin. Вони є поширені на серед тих, хто не хоче платити податки за свої кошти, але і недоліком є те, що по законодавству ви не маєте жодних прав на них і у випадку, якщо у вас їх вкрадуть, то вам нічого не повернуть. Персоніфіковані гроші, це вже ті, які зареєстровані на вас. Прикладом є гроші на банківській картці. Країни намагаються позбутися анонімних грошей, оскільки вони унеможливлюють контроль над грошима.

Переваги електронних грошей:

- зручність у транспортуванні;
- зникла необхідність у пошуках решти;
- безпека;
- зниження впливу людського фактору;
- не потрібно витрачати грошей на карбування купюр та монет.

Але одним з головних моментів, на який потрібно звернути увагу при роботі з електронними грошима є безпека. Раніше, коли відкривався магазин, то необхідно було мати лише касовий апарат та людину, яка буде його охороняти. Зараз же, коли магазин розроблює свій онлайн-магазин, то йому необхідно приділити велику увагу його безпеці, оскільки у випадку злому сайту шахраями, це може принести неабиякі збитки та магазин втратить репутацію перед своїми клієнтами. Тому компанії часто звертаються до сторонніх компаній, які надають цей функціонал, чим знімають з себе усю відповідальність, а інколи навіть економлять кошти на цьому.

Також важливим питання є те, наскільки важко підключатися до різних платіжних систем, оскільки користуватися функціоналом лише одного платіжного провайдера є не зручним, оскільки функціоналу не буде вистачати. Підключити платіжну систему це ще не все, оскільки вони кожного разу змінюються і необхідно їх підтримувати.

Основною метою магістерської дисертації було розробити агрегатор платіжних систем, до якого можна приєднувати безліч різних платіжних провайдерів та надати інтерфейс керування ними.

Задачею роботи було на основі огляду існуючих рішень розробити архітектуру додатку, запрограмувати її та описати.

Практичним використання роботи є те, що застосунок буде корисним для середнього та великого бізнесу.

У розділі 1 описується постановка задачі.

У розділі 2 досліджуються існуючі рішення.

У розділі 3 обґрунтовується вибір засобів реалізації.

У розділі 4 міститься реалізація системи.

У розділі 5 представлено інструкцію користувача.

У розділі 6 представлено стартап-проект.

У додатку А знаходиться діаграма прецедентів.

У додатку Б знаходиться структурна схема сервісу.

У додатку В описується ER-діаграма бази даних платіжного шлюзу.

У додатку Г представлено діаграму класів Payment Gateway.

У додатку Д знаходиться діаграма послідовності створення платежу у сервісі.

У додатку Е описується діаграма класів організації.

У додатку Ж представлено діаграму вибору стратегії.

У додатку И описується діаграма вибору P2P-платежу.

У додатку К представлено лістинг програми.

1 ПОСТАНОВКА ЗАДАЧІ

Основною метою дисертації було розробити агрегатор платіжних систем, до якого можна приєднувати безліч різних платіжних провайдерів та надати інтерфейс керування ними.

Одним з завдань було реалізувати сервіс платежів – Payment Gateway (надалі PG) та виплат – Payout Gateway (POG). POG має більш зрозумілий шлях використання, оскільки користувач надає номер карт або інший ідентифікатор свого рахунку та просто робимо виплату. PG має трохи більше варіантів проведення операції, але в цьому проекті є реалізований тільки перенаправлення на платіжні сторінки, бо цей варіант не потребує проходження ніяких сертифікацій.

Наступним кроком була розробка схеми маршрутизації платежів та виплат. Основною ідеєю є те, щоб адміністратор системи сам впливав на те, через які платіжні системи (PSP) буде проведено операцію, або система сама розуміла, коли потрібно провести операцію по іншому маршруту. Наприклад, якщо на балансі закінчились гроші, то немає сенсу проводити виплату по цій PSP.

Останнім важливим етапом було розробка налаштувань, для кожного підключеного аккаунту, серед яких є:

- денний та місячний оборот по виплатам та платежам окремо;
- на яку суму можна перевищити баланс, або як його називають – овердрафт;
- верхня та нижня межа, за які не може переходити баланс аккаунта.

2 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА СЦЕНАРІЇ ВИКОРИСТАННЯ СЕРВІСУ

2.1 Платіжний провайдер Stripe

Stripe[1] – це американська технологічна компанія, яка розроблює рішення для обробки електронних платежів. Вона на ринку з 2011 року та в ній працює близько 2000 людей. На рисунку 2.1 можна побачити головну сторінку Stripe.

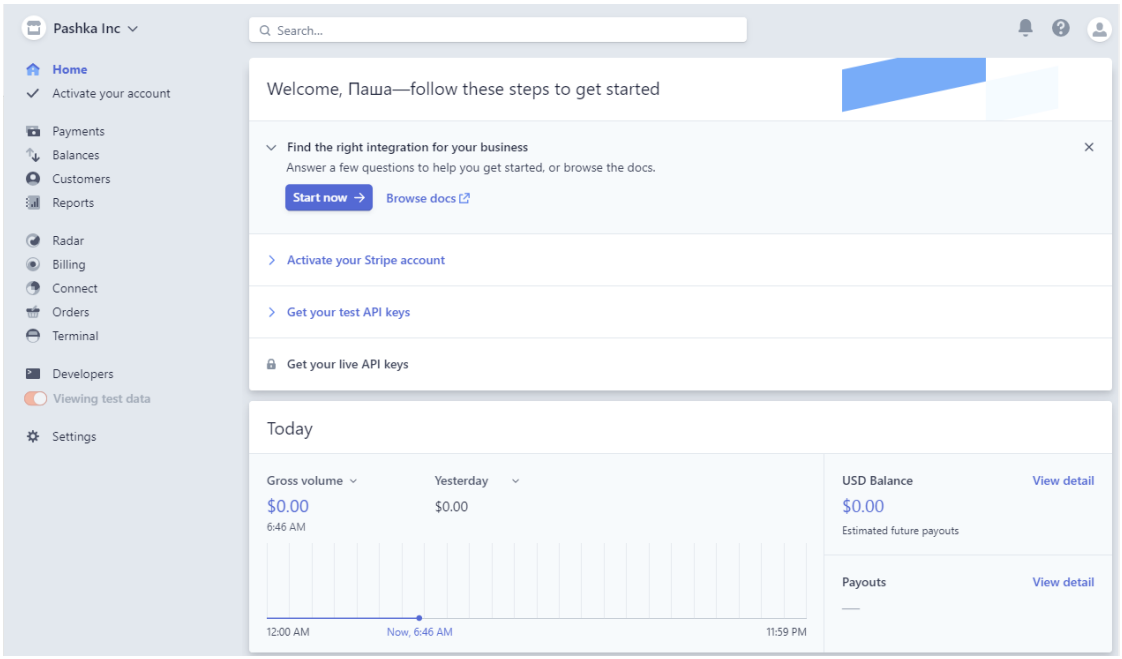

Рисунок 2.1 – Головна сторінка Stripe

Переваги Stripe:

– сучасний дизайн;

– операція поділена на декілька транзакцій, тому можна побачити весь шлях грошей;

– функція Radar, яка на основі машинного навчання та аналітики може запропонувати змінити деякі налаштування системи для кращої її роботи;

– присутній розділ “Developers”, де можна знайти останню версію API та аналітику для розробників, в якій відображається моніторинг по успішним та неуспішним запитах системи (Рисунок 2.2).

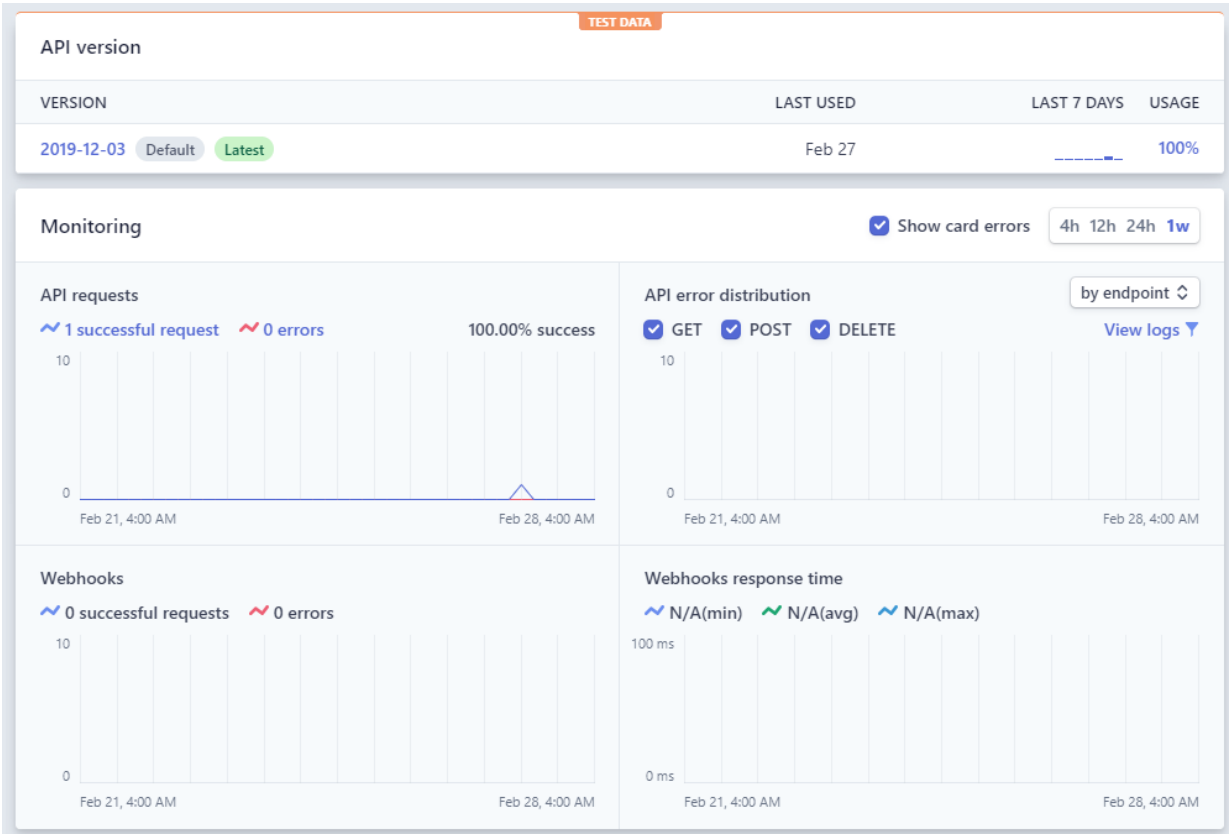

Рисунок 2.2 – Розділ “Developers”

Недоліки Stripe:

– немає схеми маршрутизації, тобто самостійно необхідно вибрати метод, яким необіхдно, щоб була проведена операція;

– ціна, оскільки за кожну успішну операцію береться комісія у розмірі 2,9% та 30 центів.

2.2 Платіжний провайдер Razorpay

Наступним для порівняння за було обрано платіжного провайдера Razorpay[2] (Рисунок 2.3):

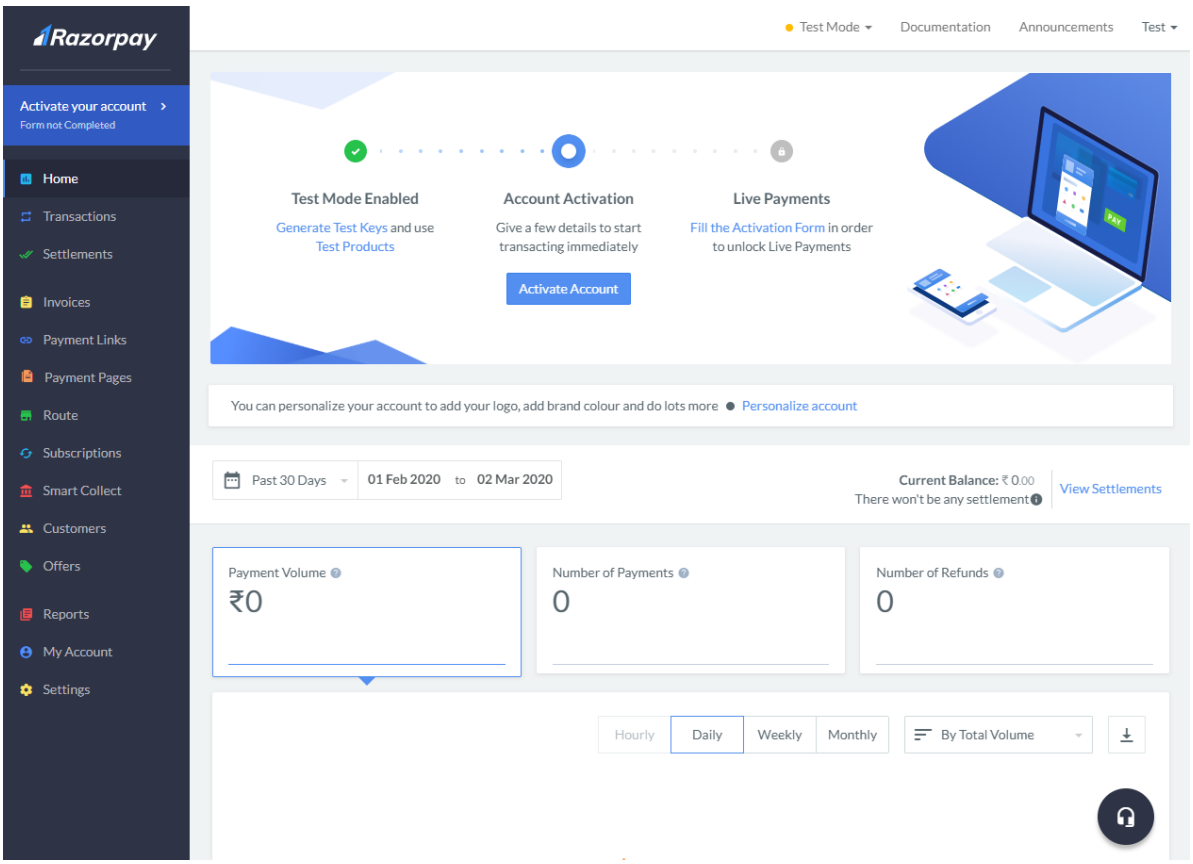

Рисунок 2.3 – Головна сторінка Razorpay

Переваги:

– сучасний дизайн;

– можливість використання коротких посилань для оплати рахунку, як зображено на рисунку 2.4;

– можливість збереження та дублювання рахунку для подальшого повторного використання;

– можливість створення своїх особистих HPP (Hosted Payment Page).

Недоліки:

– система працює лише в Індії.

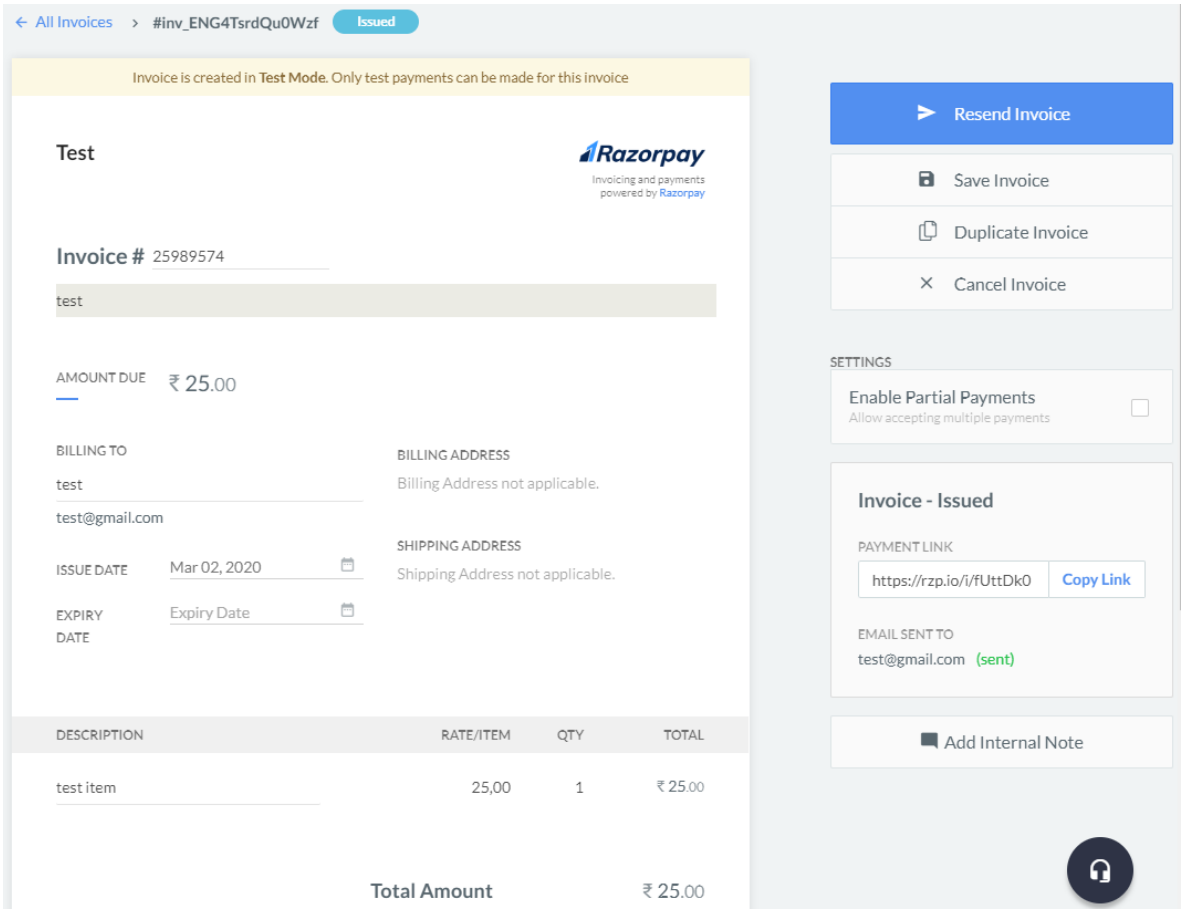

Рисунок 2.4 – Сторінка створеної операції у системі Razorpay

2.3 Сценарії використання сервісу

На додатку А можна побачити діаграму прецедентів, тобто які дії користувач та клієнт можуть робити.

User (користувач) – це людина, яка хоче витратити чи отримати кошти. Наприклад, коли людина хоче поповнити рахунок телефону, то вона створює платіж, який піде на рахунок телефонного оператора. Основні можливості користувача:

- створення платежу;
- створення виплати;
- перенаправлення на HPP.

В даному сервісі реалізовано тільки перенаправлення на сторінку платіжної системи оскільки це не потребує проходження різних сертифікацій, які необхідні для того, щоб зберігати деякі карточні або інші дані.

Як правило, виплати проходять набагато швидше платежів оскільки одразу йде запит на платіжну систему після створення, а у випадку з платежами, сервіс просто перенаправляє на сторінку оплати і більше нічого від нього не залежить, окрім очікування на зміну статусу операції. Користувач може просто залишити сторінку та відійти кудись.

Client (клієнт, тобто адміністратор) має доступ до більшої кількості функцій. Завдяки налаштувань, які він виставить буде залежить, котрим шляхом та яку комісію заплатить користувач.

- налаштування організації;
- створення платежів;
- створення виплат;
- налаштування для курсів валют;
- маршрутизація (RDP) для платежів та виплат;
- використання P2P-платежів;
- налаштування доступів до сервісу.

Висновки до розділу 2

Було проаналізовано найпопулярніші аналогічні сервіси та оброблено всі переваги і недоліки. Серед переваг було виділено те, що:

- операції поділено на декілька транзакцій;
- зрозуміле і просте API для розробників.

А серед основних недоліків, яких необхідно буде позбутися:

- немає маршрутизації для операцій;
- розроблення системи під один регіон.

Було визначено ролі системи та основні їх функції.

3 ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ

3.1 Обґрунтування вибору мови програмування

При виборі мови програмування необхідно чітко розуміти, які задачі повинна вона виконувати. Важливим критерієм є взаємодія та підтримка з базами даних, оскільки сервісу потрібно буде оброблювати велику кількість інформації та залишатися стабільним. Ще один аспект це веб орієнтованість мови програмування, оскільки сервіс буде працювати лише через Інтернет, а це все знижує кількість мов. І останнім критерієм є популярність мови через розробників, щоб не було проблем знайти когось для розробки та підтримки сервісу.

Тож зважаючи на все вищесказане для виконання магістерської дисертації було обрану мову PHP через ряд деяких переваг:

– велика кількість розробників застосовують цю мову програмування, отже не буде проблемою знайти людей на розробку та підтримку;

– PHP достатня стара мова програмування, тому має багато довідкового матеріалу;

– легкий доступ до бази даних;

– мова PHP може використовуватись на різних платформах змінивши лише декілька конфігураційних файлів;

– безперечно, одна з найпопулярніших мов в світі для програмування веб- застосунків.

Але і варто згадати про декілька недоліків цієї мови програмування:

– необхідно бути обережним з обчисленнями, оскільки PHP не закінчує роботу програми, якщо, наприклад, поділити на нуль, а просто присвоїть йому значення false, що може трактуватися як 0;

– буває важко дотримуватися стандартів написання коду, що в майбутньому призведе до поганої читабельності коду;

– недосконалий набір роботи з виключеннями.

Проблеми з безпекою існують в файлі php.ini – головному конфігураційному файлі PHP. Там зберігаються параметри для всіх платформ і там їх сотні та не всі з них добре описані. Але проблема не в тому, що їх багато, а в тому що зміна одного параметру може вплинути на інші, це призводить до помилок, які важко виявити.

Мова має велику документацію та велику кількість книжок в яких можна знайти статті з описами того, як зробити свій застосунок безпечнішим.

Однією з гарних сторін вибору цієї мови програмування є те, що вона і далі розвивається. Вже під час написання цієї роботи виходить нова, восьма версія, у якої багато нового функціоналу та збільшення продуктивності.

Часто можна почути, що мову PHP вважають повільною[3]. Це дійсно так, якщо невміло нею користуватися, а якщо, наприклад, використовувати для неї додатки написані на більш низкорівневих мовах програмування або використовувати систему кешування, то вона може бути однією із найшвидших.

На рисунку 3.1 можна побачити наскільки швидшою стає мова з кожною версією. На графіку видно, що версія 7.2 на 22 відсотки швидша ніж версія 7.0[4], а між ними пройшло не так і багато часу.

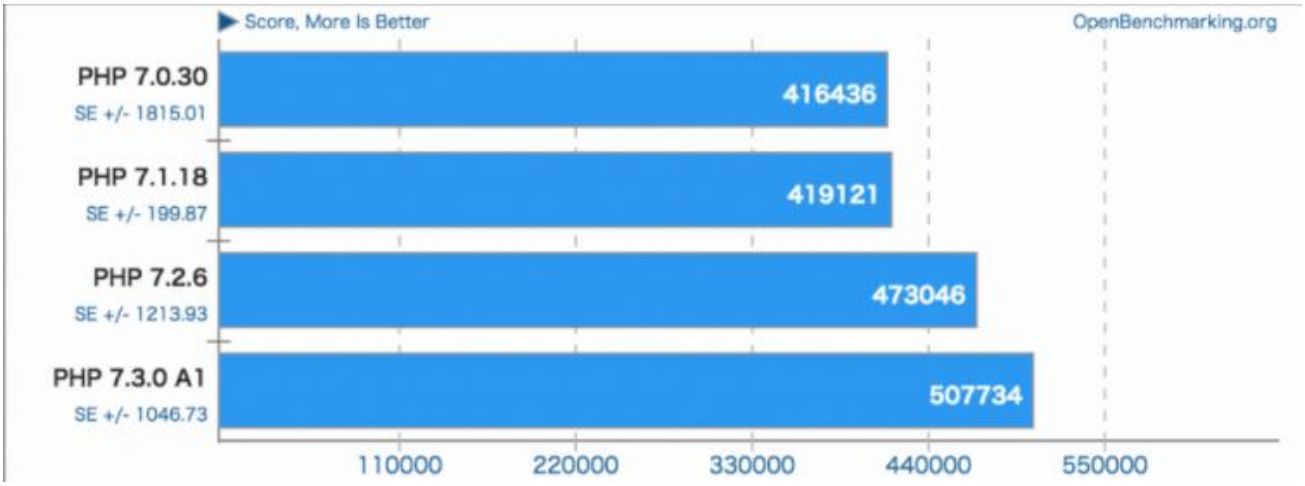

Рисунок 3.1 – Приріст швидкості роботи мови PHP

3.2 Фреймворк

Для розробки сервісу було використано фреймворк Symfony[5]. Symfony – це безкоштовний фреймворк з відкритим вихідним кодом, перша версія якого була випущена в 2005 році, що робить його найстаршим серед популярних нині PHP фреймворків та свідчить про те, що йому вдається кожного разу відстоювати свої лідерські позиції у цій галузі. Фреймворк – це шматок програмного забезпечення, який можна використовувати для свого додатку, бо він має шляхи вирішення ряду повсякденних задач з якими стикається програміст.

Також це є гарним варіантом оскільки не треба перейматися за такі речі, як взаємодія з базою чи автентифікація користувачів, ці речі є базовими для кожного сервісу і вже давно реалізовані не тільки в обраному фреймворку, а і в усіх інших. Якщо би довелося самостійно розроблювати даний функціонал, то знадобилося набагато більше часу на це і не було би ніяких гарантій, що десь не допущено помилку, яка може скомпрометувати сервіс.

Також перевагами є:

– довідкова база фреймворку;

– шаблон MVC;

– можливість розділяти конфігураційні файли на різні оточення, а отже можна тестувати сервіс з новими параметрами і не ризикувати за збій сервісу вже в реальній роботі;

– використання ORM (Object-Relational Mapping) – технологія для створення віртуально об’єкту бази даних – Doctrine.

Doctrine – найпопулярніша ORM та найпопулярніший спосіб з’єднання з базою даних у мові PHP. На рисунку 3.2 зображено принцип роботи ORM. Вона як прошарок між кодом та базою даних.

Рисунок 3.2 – Принцип роботи ORM[6]

Ключовою характеристико є те, що вона використовує власний об’єктно-орієнтований діалект SQL – DQL (Doctrine Query Language). За допомогою її можна використовувати базу даних, як об’єкт.

3.3 База даних

Як СУБД було обрано PostgreSQL через такі переваги:

– підтримка типу даних JSON та багато інших;
– розмір даних компонентів бази даних (таблиця 3.1);
– цілісність даних;
– гарно та детально описана специфікація та велика кількість інформації в Інтернеті.

Таблиця 3.1 – Розмір даних компонентів бази даних [7]

Максимальний розмір бази даних	Необмежено
Максимальний розмір таблиці	32 TB
Максимальний розмір строки	1.6 TB
Максимальний розмір поля	1 GB
Максимальна кількість стрічок в таблиці	Необмежено
Максимальна кількість стовбців в таблиці	250-1600 в залежності від типу стовпця
Максимальна кількість індексів в таблиці	Необмежено

На перших етапах база даних буде не сильно навантажено, але після того, як сервіс отримає багато клієнтів вона буде наповнюватися терабайтами даних, а PostgreSQL має великий набір інструментів, щоб можна було очищати сміття, очищати старі транзакції та інше.

Однією з великих переваг PostgreSQL є великий вибір індексів, які підходять до різних типів задач. Такою задачею в даній магістерській роботі стало пошук по великому об’єму даних, оскільки при великому трафіку системи через пару років база може важити не один терабайт даних. Детально дослідивши це питання було обрано вибрати індекс GIN[8] – Універсальний інвертований індекс. Він розділяє дані на слова та зберігає їх в себе зі списком місць де вони знаходяться. Приклад індексу можна побачити на рисунку 3.3.

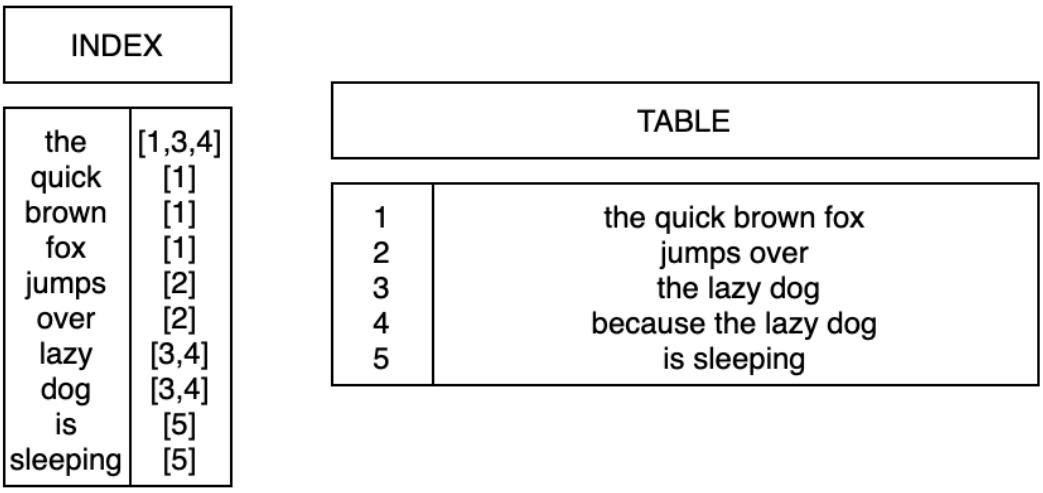

Рисунок 3.3 – Приклад роботи індексу GIN

Коли йде пошук по декільком слів то достатньо лише знайти перше, а після того можна скористатися індексом і виключити зі списку ті, які слова не доступні. Також слід зазначити, що роботу цього індексу можна пришвидшити, наприклад параметром maintenance_work_mem, в той час як схожі за призначенням індекси, наприклад GIST (Generalized Search Tree) програють йому у цьому аспекті.

3.4 API

Для взаємодії з сервісом було обрано специфікацію JSON:API[9]. JSON: API – це специфікація того, як клієнт повинен відправляти запити, отримувати відповідь, працювати з ресурсами та як сервер повинен реагувати та відповідати на запити клієнтів. Вона найкраще підходить у тих випадках, коли необхідно описати саме сутність системи. Наприклад, коли створюється аккаунт або він змінюється, то при успішному запиті сервіс повертаємо саму сутність. У випадках при яких сервіс повинен повернути помилку або просто щось на кшталт «OK» використовуються звичайні відповіді у форматі JSON.

У магістерській дисертації було використано пакет neomerx/json-api[10]. Даний пакет має гарне покриття коду тестами та має велику кількість завантажень, що свідчить про його якість. Також цей пакет реалізую останню

версію JSON:API, тож можна не переживати за те, що буде використана не остання технологія. Не слід забувати, що під час роботи з великою кількістю даних одним з головних критеріях є їх фільтрація. Ось приклад посилання з фільтрами «/payments?filter[status]=success&filter[test_mode]=false» і фільтрів може буди необмежена кількість.

На рисунку 3.4 можна побачити приклад структури відповіді обраної специфікацій.


```
{
  "links":{
    "self":"http://example.com/articles",
    "next":"http://example.com/articles?page[offset]=2",
    "last":"http://example.com/articles?page[offset]=10"
  },
  "data":[
    {
      "type":"articles",
      "id":"1",
      "attributes":{
        "title":"Example",
        "description":"example"
      },
      "relationships":{
        "author":{
          "links":{
            "self":"http://example.com/articles/1/relationships/author",
            "related":"http://example.com/articles/1/author"
          },
          "data":{
            "type":"people",
            "id":"9"
          }
        },
        "links":{
          "self":"http://example.com/articles/1"
        }
      }
    }
  ]
}
```


Рисунок 3.4 – Приклад відповіді JSON:API

Links – об’єкт для того, щоб зберігати різні посилання для пагінації, він може змінюватися в залежності від її типу. Data – об’єкт, де зберігається вся необхідна інформація про сутність і ось його поля:

– «type» – тип сутності до якої йде запит;

– «id» – унікальний ідентифікатор сутності;

– «attributes» – місце, де зберігається усі поля сутності;

– «relationships» – зберігає в собі посиланні та ідентифікатори сутностей, які пов’язані з нею.

Звісно, що не можна щоб доступ до API сервісу був у всіх, оскільки велика ймовірність того, що цим можуть скористатися шахраї. При виборі способу автентифікації було приділено особливу увагу збереженню балансу між безпекою та легкістю використання даного способу. Цьому було приділено таку увагу оскільки були випадки, коли ті ж платіжні системи робили занадто складну автентифікацію і це призводило до великих затримок під час розробки взаємодії. Також у випадку, коли потрібно буде щось змінити або трапиться помилка, на її знаходження та усунення витратиться більше часу.

Ось основні схеми автентифікації:

– Basic – при якій в заголовках передається Autorization у base64-encoded вигляді;

– Digest – при якому сервер надсилає унікальне значення nonce, а браузер передає хеш пароля користувача;

– NTLM – використовується загалом для користувачів у веб-додатках;

– Negotiate – оснований на технології Single Sign-On[11], але безпеку гарантує тільки якщо сервер і клієнт знаходяться на доменах Windows.

На основі цих даних та дослідженню усіх способів взлому таких схем було обрано Basic authentication[12]. На рисунку 3.5 можна побачити приклад його роботи. Спочатку при неавтентифікованому запиті на сервер користувачу віддається 401 код, потім після того як він надіслав у заголовку свій логін та пароль йому віддається 200 код і вже на всі подальші запити у заголовках браузер буде передавати дані у заголовках. Але не слід забувати, що браузер час від часу видаляє старі дані та і сам користувач може ненароком їх видалити, тому треба буди готовим повторити цю процедуру заново для з’єднання з сервером.

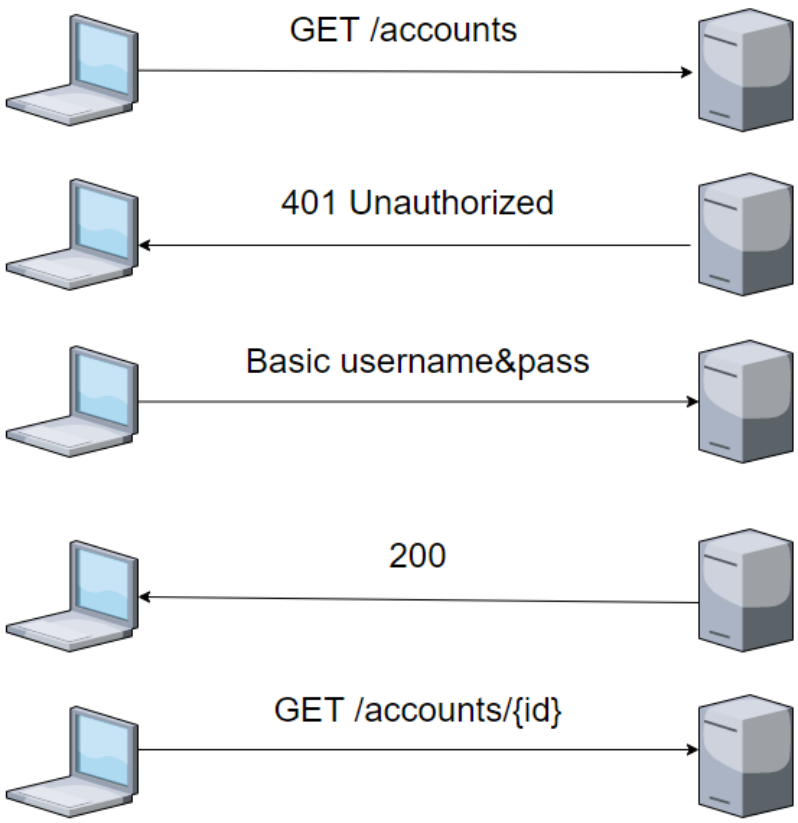

Рисунок 3.5 – Принцип роботи Basic authentication

Можливо в подальшому можна буде змінити тип автентифікації, але на даному етапі він повністю задовольняє вимогам сервісу.

Висновки до розділу 3

Звернувши увагу на все вище сказане, обраний стек технологій повністю відповідає всім вимогам, адже безпека, швидкість та масштабування є основними перевагами кожного сервіса. Графічно продемонстровано наскільки мова збільшує у швидкості при кожній новій версії. Також великий плюс до вибору цього стеку технологій є велика база інформації мови PHP та інших технологій, що будуть використані у додатку. Також вказано, що мова і далі розвивається, отже можна не перейматися за її оновлення.

Завдяки використанню Symfony та Doctrine буде мінімізована можливість виникнення помилок на етапі архітектури, оскільки вони надаються свої інтерфейси, які вже перевірені роками та мають широку аудиторію.

Описано переваги обраної СУБД, а саме велику кількість індексів та зберігання значного об’єму інформації. Описано, як клієнти будуть взаємодіяти с сервісом та обрано бібліотеку, яка буде за це відповідати.

Проведено аналізів основних схем автентифікації запитів, обрано та детально описано схема, яка було застосована у сервісі.

4 РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

4.1 Структура коду та методи його покращення

Перевагою використання фреймворку є те, що він надає своє структуру розміщення папок та файлів як зображено на рисунку 4.1.

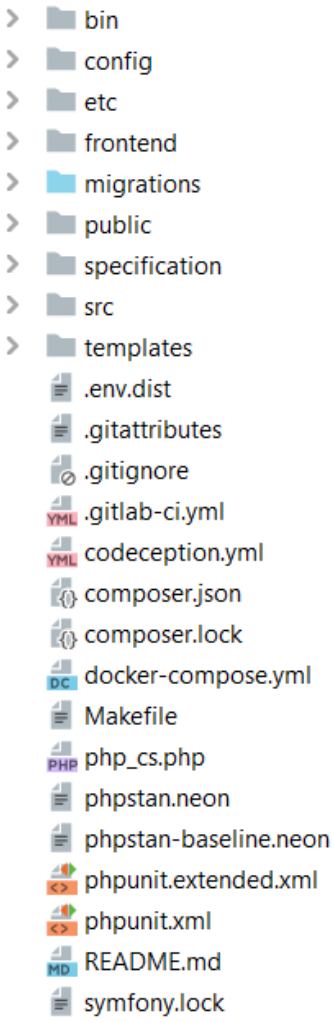

Рисунок 4.1 – Файлова структура проекту

В папках config та etc розміщенні основні конфігураційні файли системи, там налаштовується підключення до бази даних та сервіс для запитів, також при необхідності зберігання файлів можна підключити сервіс для цього. Папка frontend містить файли для відображення контенту та їх перекладами, migrations містить SQL запити до бази даних для зміни контенту. Документація swagger[13] зберігається в папці specification, а в папці src знаходиться весь основній код

додатку. Також купа інших файлів для composer, статичного аналізатора коду, тестів та інших допоміжних файлів фреймворку та всього проекту.

Для того, щоб контролювати те, що код має один вигляд було обрано пакет friendsofphp/php-cs-fixer[14], також від розробників Symfony. На рисунку 4.2 можна побачити файл для налаштування. Наприклад перед такими виразами, як return, try або continue потрібно додавати пусту строку.

```
$finder = PhpCsFixer\Finder::create()
    ->in('src')
    ->exclude('Paycore/Core/FosUserBundle')
    ->notName('extension-generated-templates.php');

return PhpCsFixer\Config::create()
    ->setRules([
        'blank_line_before_statement' => [
            'statements' => [
                'break',
                'continue',
                'return',
                'throw',
                'try'
            ]
        ],
        '@PSR2' => true,
        '@Symfony' => true,
        '@Symfony:risky' => true,
        '@PHP70Migration:risky' => true,
        '@PHP71Migration:risky' => true,
        'concat_space' => [
            'spacing' => 'one',
        ],
        'strict_param' => true,
        'mb_str_functions' => true,
        'declare_strict_types' => true,
        'array_syntax' => [
            'syntax' => 'short'
        ],
```

Рисунок 4.2 – Файл налаштування php-cs-fixer

Досить важко власноруч контролювати код на працездатність, оскільки кожен неіснуючий метод може завадити коректним виконанням роботи сервісу. Для того, щоб цього не допускати було рішення використати статичний аналізатор коду. Проаналізувавши всі аналізатори коду для мови PHP було обрано PHPStan[15], він має декілька рівнів перевірки, тому можна обрати той, який вас задовольняє.

4.2 Архітектура системи

У додатку Б наведено структурну діаграму даного сервісу. Вона поділена на окремі сектори, які пов’язані між собою, але не всі з них є частиною даного сервісу.

Сектор «Платіжні провайдери» містить основу всього – платіжні системи, без них не було б необхідності у даному сервісі. Блок PSP означає, що на це місце можна підставити, тобто підключити, будь-яку платіжну систему у якої є можливість взаємодії по API.

Наступний сектор це Connectors, тобто місце де відбувається вся взаємодія з платіжними системами. Як можна бачити на додатку Б для кожного конектору є своя платіжна система, але як виняток з правил для двох PSP може бути реалізовано один конектор, якщо вони мають однаковий API.

Далі йде основа сервісу, місце де контролюється створення усіх платежів та виплат, проводиться ходження по балансах аккаунтів та зберігаються майже всі налаштування сервісу. Весь функціонал поділений між трьома блоками:

– Provider Hub – місце, де відбувається взаємодія з аккаунтами платіжних систем;

– Payment Gateway (PG) – відповідає за створення платежів та обробки їх статусів;

– Payout Gateway (POG) – теж саме, що і PG тільки для виплат.

Також можна побачити, що вони не зовсім самостійні оскільки між ними також є комунікація. І один з найголовніших місць сервісу є API взаємодія. Немає ніякої однієї вхідної точки до сервісу, запити йдуть до необхідного блоку.

У додатку В та додатку Г наведені ER-діаграму бази даних платіжного шлюзу та діаграму класів платіжного шлюзу. Перша сутність, з якої починається підключення це MerchantAccount, вона має:

– id – унікальний ідентифікатор;

– provider – ознака платіжного провайдера (наприклад liqpay);

– currency - валюта;

– payload – місце, де зберігаються ключі доступу до платіжної системи;

– externalId – ідентифікатор платіжної системи.

Для кожного платіжного методу існує свій PaymentService, який має усю необхідну інформацію про нього:

– code – назва сервісу, наприклад payment_card_uah_hpp, правило формування назви можна подивитися на рисунку 4.2;

– fields – поля, які необхідно передати для цього сервісу, наприклад номер карточки;

– currency – валюта;

– amountMin – нижня границя суми;

– amountMax – верхня границя суми.

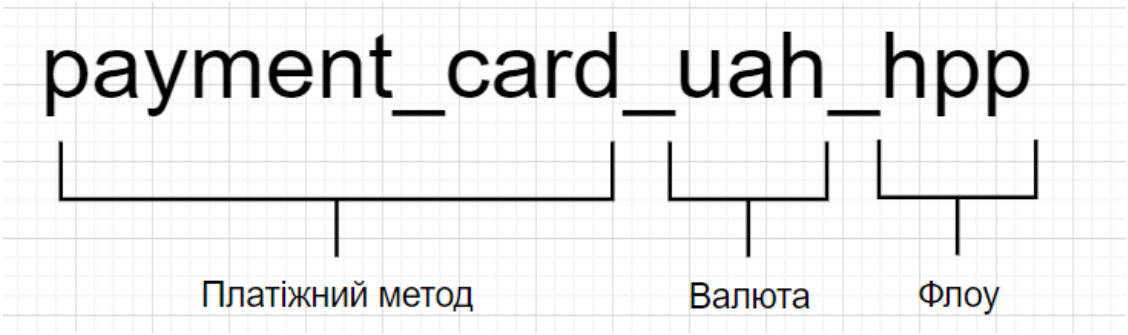

Рисунок 4.2 – Правило формування назви платіжного сервісу

Для кожного аккаунту створюється один або декілька платіжних маршрутів, це залежить від того, скільки платіжних методів підтримую провайдер (наприклад в LiqPay їх 6: платіжна карта, термінал, liqpay_wallet та інші). Маршрут має такі поля:

– provider – ознаку платіжного провайдера;

– merchantAccount – посилання на MerchantAccount;

– PaymentService який буде використовуватися;

– enabled – ознака, чи включений маршрут;

– amountMin/amountMax – ліміти на суми по маршруту;

– feeFix – фіксоване значення комісії;

– feeRate – відсоток комісії;

– feeMin – нижня границя комісії;

– feeMax – верхня границя комісії;

– p2p – можливість виконувати P2P-операції;

– p2pOptions – місце збереження налаштувань для роботи P2P-платежів.

Далі в нас йде сутність самої платіжної операції – PaymentOperation:

– route – посилання на PaymentRoute, через який був створений платіж;

– amount – сума платежу;

– fee – комісія;

– currency – валюта

– status – статус платежу у даний момент часу;

– resolution – причина, чому платіж не відбувся;

– statement_payload – вся додаткова інформація, шо нам вдалось дістати з відповіді PSP;

– created – дата створення операції;

– description – опис операції;

– referenceId – унікальний ідентифікатор платежу в системі клієнтів сервісу;

– externalId – ідентифікатор операції на стороні PSP;

– externalStatus – статус платежу в PSP;

– externalResolution – додатковий код PSP;

– externalMessage – пояснення до платежу від PSP.

– exchageRate – курс валюти за якою була зроблена операція.

Є ще одна сутність – PaymentStatement:

– operation – посилання на PaymentOperation, до якого він відноситься;

– amount – сума платежу;

– fee – комісія;

– currency – валюта

– status – статус платежу у даний момент часу;

– resolution – причина, чому платіж не відбувся;

– payload – вся додаткова інформація, шо нам вдалось дістати з відповіді PSP;

– created – дата створення операції;

– description – опис операції;

– referenceId – унікальний ідентифікатор платежу в системі клієнтів сервісу;

– externalId – ідентифікатор операції на стороні PSP;

– externalStatus – статус платежу в PSP;

– externalResolution – додатковий код PSP;

– externalMessage – пояснення до платежу від PSP.

PaymentStatement можна вважати, як випискою з PSP в конкретний момент часу, це і є його головною відмінністю від PaymentOperation.

OrganizationAware – інтерфейс для того, щоб було зрозуміло, які сутності відносяться до конкретної організації, яка в свою чергу створюється для кожного клієнта. Це відображає те, що PaymentService не є приналежністю якоїсь організації, а є налаштуванням та сутностями усієї системи.

Слід зазначити, що система є майже повністю консистентною, тобто сутності системи не видаляються, а відключаються чи архівуються. Це дає переваги в тому, що данні не будуть суперечити один одному та можна розділити данні у разі необхідності.

POG немає сенсу описувати так само, оскільки він майже повністю за винятком якихось речей повторює PG, але нижче буде наведено опис бази даних деяких сутностей. У таблиці 4.1 наведено опис таблиці deposit_accounts.

Таблиця 4.1 – Таблиця deposit_accounts

Назва колонки	Тип даних	Призначення
id	varchar(64)	Унікальний ідентифікатор аккаунта

Назва колонки	Тип даних	Призначення
external_id	varchar(64)	Ідентифікатор аккаунту на стороні платіжної системи
amount	numeric(19,8)	Баланс
overdraft	numeric(19,8)	Сума, на яку можна перевищувати баланс
amount_limit_lower	numeric(19,8)	Сама, нижче якої не може опуститися баланс
currency	varchar(16)	Валюта
name	varchar(64)	Назва
description	text	Опис
auto_updating	bool	Показує чи є у платіжної системи можливість віддавати баланс
test_mode	bool	Тестовий режим
status	varchar(64)	Показу, чи доступний зараз аккаунт
turnover	numeric(19,8)	Скільки коштів можна виплатити з цього аккаунту за день
created	timestamp(3)	Дата створення
updated	timestamp(3)	Дата останнього оновлення
organization	varchar(64)	Посилання на організацію
payment_provider	varchar(64)	Код платіжної системи

У таблиці 4.2 наведено опис таблиці payout_routes.

Таблиця 4.2 – Таблиця payout_routes

Назва колонки	Тип даних	Призначення
id	varchar(64)	Унікальний ідентифікатор маршруту
enabled	bool	Ознака включеного маршруту
archived	bool	Ознака архівованого маршруту
external_id	varchar(64)	Ідентифікатор маршруту на стороні платіжної системи
test_mode	bool	Ознака, чи маршрут є тестовим
service_fields	json	Поля, які необхідні для даного сервісу
fee_min	numeric(19,8)	Мінімальне значення комісії
fee_max	numeric(19,8)	Максимальне значення комісії
fee_rate	numeric(19,8)	Відсоток комісії
fee_fix	numeric(19,8)	Фіксована сума комісії
amount_min	numeric(19,8)	Мінімальна сума виплати
amount_max	numeric(19,8)	Максимальна сума виплати
created	timestamp(3)	Дата створення
updated	timestamp(3)	Дата останнього оновлення
organization	varchar(64)	Посилання на організацію
payment_provider	varchar(64)	Код платіжної системи
p2p	bool	Ознака, чи підтримує маршрут P2P

Назва колонки	Тип даних	Призначення
deposit_account	varchar(64)	Посилання на аккаунт
service	varchar(64)	Посилання на PayoutService

4.3 Взаємодія з PSP

На рисунку 4.3 можна побачити файлову структуру платіжної системи.

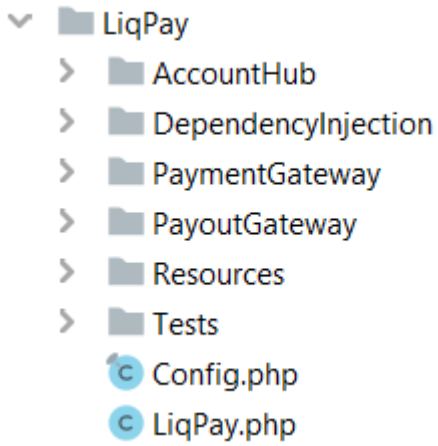

Рисунок 4.3 – Файлова структура платіжної системи

Взаємодія з платіжними системами є одним з найважливіших місць роботи оскільки від того, як гарно буде налагоджений зв’язок з ними буде залежити робота всього сервісу.

Опис кожної директорії:

– AccountHub – місце де відбувається вся взаємодія з API, міститься інформація про те, як використовувати та перевіряти ключі доступу до платіжки;

– PaymentGateway/PayoutGateway – відповідно взаємодія з платіжним шлюзом та шлюзом виплат;

– Resources – містить в собі всі конфігураційні файли та приклади відповідей від платіжної системи для того, щоб в подальшому використовувати їх в тестах;

– Tests – тести платіжної системи;

– DependencyInjection, Config, LiqPay(назва змінюється в залежності від назви платіжної системи) – допоміжні файли і майже не змінюються в залежності від PSP.

На рисунку 4.4 можна побачити статуси платежів, назви яких були обрані проаналізувавши найпопулярніші статуси платіжних систем та аналогових рішень та представлені саме так, щоб вони були зрозумілі і мали тільки одне конкретне значення.

```
public const CREATED = 'created';
public const PENDING = 'pending';
public const SUCCESS = 'success';
public const DECLINED = 'declined';
public const FAILED = 'failed';
public const UNKNOWN = 'unknown';
```

Рисунок 4.4 – Статуси платежів

Основне, що треба розуміти при взаємодії, це те, що необхідно лише правильно інтерпретувати відповіді платіжної системи і перенести їх на модель сервісу.

А вже на рисунку 4.5 можна побачити приклад частини карти статусів платіжної системи LiqPay. Це приклад того, як сервіс розуміє статуси інших платіжних систем, потрібно гарно вивчити статуси інших платіжних систем, оскільки кожен невірний статус може призвести до фінансових втрат. Останній статус «unknown» (невідомий) відповідаю за ті випадки, коли нам прийшов статус, який не був описаний в документації або по іншим причинам не доданий до сервісу. При цьому статусі зупиняється обробка платежу оскільки є ризик втрати коштів. Клієнту необхідно дізнатися у платіжної системи причини такої відповіді і допровести платіж власноруч.

```
public static $map = [
    [
        self::EXTERNAL_CODE => TransactionStatusCode::SUCCESS,
        self::INTERNAL_CODE => StatementStatus::SUCCESS,
    ],
    [
        self::EXTERNAL_CODE => TransactionStatusCode::ERROR,
        self::INTERNAL_CODE => StatementStatus::FAILED,
    ],
    [
        self::EXTERNAL_CODE => TransactionStatusCode::PROCESSING,
        self::INTERNAL_CODE => StatementStatus::PENDING,
    ],
    [
        self::EXTERNAL_CODE => TransactionStatusCode::PREPARED,
        self::INTERNAL_CODE => StatementStatus::CREATED,
    ],
```

Рисунок 4.5 – Карта статусів

Карти використовуються не лише для статусів, а і для резолюшенів, маршрутів та іноді для валют, оскільки в сервісі використовуються код валюти ISO 4217[16], а в інші платіжки можуть використовувати інші формати.

Першим, що потрібно зробити для користувача це ввести ключі доступу від PSP і після того як пройшли звичайні перевірки на довжину полів, тип та регулярний вираз необхідно перевірити, що це справжні ключи. Для цього є декілька варіанті перевірки, одним з найпопулярніших є запит на отримання балансу і у випадку його отримання можна зрозуміти, що ключі дійсно справжні. Але наприклад для LiqPay ця перевірка є не зовсім підходящою, тому там використовується запит на отримання статусу платежу по випадково згенерованому ідентифікатору платежу. Якщо ключі доступу дійсно правильні, то буде отримана відповідь як на рисунку 4.6, відповідь означає, що платежу немає, а якщо ключі неправильні то відповідь буде відповідна.

Рисунок 4.6 – Відповідь платіжки на перевірку статусу неіснуючого платежу

Після того, як сервіс переконався, що всі доступи є і MerchantAccount вже створений, то переходимо до створення PaymentRoute. Деякі платіжні системи надаються окремий запит на отримання їх способів оплати, але для більшості є власноруч зібрана карта маршрутів по якій вони і створюються.

На рисунку 4.7 продемонстровано метод для створення аккаунтів для виплат.

```
public function getDepositAccounts(): array
{
    $depositAccountDtos = [];
    $currencies = $this->getCredentials()->getCurrencies();

    foreach ($currencies as $currency) {
        $balance = $this->getBalance($currency);
        $depositAccountDto = new DepositAccountDto(
            new Currency($currency),
            \sprintf( format: '%s_%s', $this->getCredentials()->getPublicKey(), $currency)
        );

        $money = AmountFormatter::formatToMoney((string) \round($balance['b2c_current_balance'], precision: 2), $currency);
        $depositAccountDto->setBalance($money->getAmount());
        $depositAccountDtos[] = $depositAccountDto;
    }

    return $depositAccountDtos;
}
```

Рисунок 4.7 – Метод створення аккаунтів для виплат

Окрім того, що саме тут вирішується, які аккаунти будуть створені та присвоєння, цей метод займається тим, що відправляє запит на платіжну систему, щоб дістати баланс аккаунту або іншу корисну інформацію, звісно, тільки для тих в кого присутній даний функціонал. Також для того, щоб кожен раз оновлювати

баланс, було створено команду deposit-account:balance:sync, яка запускається кожну хвилину і оновлює всі аккаунти, які мають такі можливості.

Бувають випадки, коли платіжні системи можуть віддавати не одна значення балансу, а декілька. Наприклад, інколи можна отримати баланс по декільком різних банків або один баланс буде активний, а інший буде доступний згодом. У таких випадках ситуація вирішується на рівні конеткора, в деяких випадках ці значення можуть просто додавати, а в інших використовувати лише одне з них за потреби клієнта.

На рисунку 4.8 можна побачити список полів з поясненнями, які необхідно відправити, щоб створити платіж.

Parameter	Required	Type	Description
version	Required	Number	Версія API. Поточне значення - 3
public_key	Required	String	Публічний ключ - ідентифікатор магазину. Отримати ключ можна в налаштуваннях магазину
action	Required	String	Тип операції. Можливі значення: pay - платіж, hold - блокування коштів на рахунку відправника, subscribe - регулярний платіж, paydonate - пожертва, auth - предавторізація картки
amount	Required	Number	Сума платежу. Наприклад: 5, 7.34
currency	Required	String	Валюта платежу. Можливі значення: USD, EUR, RUB, UAH
description	Required	String	Призначення платежу.
order_id	Required	String	Унікальний ID покупки у Вашому магазині. Максимальна довжина 255 символів.
expired_date	Optional	String	Час до якого клієнт може оплатити рахунок за UTC. Передається в форматі 2016-04-24 00:00:00
language	Optional	String	Мова клієнта ru, uk, en
paytypes	Optional	String	Параметр в якому передаються способи оплати, які будуть відображені на чекауті. Можливі значення apay - оплата за допомогою Apple Pay, gpay - оплата за допомогою Google Pay, card - оплата карткою, liqpay - через кабінет liqpay, privat24 - через кабінет приват24, masterpass - через кабінет masterpass, moment_part - розстрочка, cash - готівкою, invoice - рахунок на e-mail, qr - сканування qr-коду. Якщо параметр не переданий, то застосовуються настройки магазину, вкладка Checkout.
result_url	Optional	String	URL у Вашому магазині на який покупець буде переадресовано після завершення покупки. Максимальна довжина 510 символів.
server_url	Optional	String	URL API в Вашому магазині для повідомлень про зміну статусу платежу (сервер -> сервер). Максимальна довжина 510 символів. Детальніше
verifycode	Optional	String	Можливе значення Y. Динамічний код верифікації, генерується і повертається в Callback. Так само згенерований код буде переданий в транзакції верифікації для відображення у виписці по картці клієнта. Працює для action = auth.

Рисунок 4.8 – Поля, які необхідні для створення платежу в LiqPay

На цьому етапі можна рахувати, шо все повністю підключено та можна переходити до створення операцій. Існують декілька підходів по створенню операцій у платіжних системах за допомогою їх платіжних сторінок:

– відправити форму з даними на статичне посилання;

– відправити запит на створення посилання для оплати, а вже потім перенаправити користувача туди;

– інші варіанти під час яких необхідно відправити декілька запитів, щоб платіж відбувся.

Після того, як всі необхідні поля зібрані, то сервіс надсилає ці данні за посиланням https://www.liqpay.ua/api/3/checkout і у випадку коректності всіх даних, перед користувачем відкриється платіжна сторінка з тим платіжним методом який він вибрав (мається на увазі, який PaymentRoute був використаний).

Наступним кроком після перенаправлення користувача є дізнавання статусу платежу на стороні PSP. На рисунку 4.9 можна побачити приклад, як після того, як відправили запит на отримання статусу всі данні переносяться на модель сервісу.

```php
$externalStatus = $data['status'] ?? null;
$externalResolution = $data['err_code'] ?? null;

$resolution = null === $externalResolution ? PaymentResolution::ok() : PaymentResolutionMapper::fromExternalCode($externalResolution);
$status = PaymentStatusMapper::fromExternalStatusAndResolutionCode($externalStatus, $externalResolution);

$paidMoney = self::formatMoney($data['amount'], $data['currency']);

$dto = new PaymentStatementDto(
    $type,
    $status,
    $resolution,
    $paidMoney,
    (string) $data['payment_id']
);

$dto
    ->setOperationId($data['order_id'])
    ->setAmount($paidMoney)
    ->setMethodAmount($paidMoney);

if (isset($data['receiver_commission'])) {
    $dto->setMethodOutFee(self::formatMoney($data['receiver_commission'], $data['currency']));
}

$dto
    ->setTestMode( testMode: 'sandbox' === $externalStatus)
    ->setCurrencyExternalCode($data['currency'])
    ->setExternalStatusCode($externalStatus)
    ->setExternalResolutionCode($externalResolution)
    ->setExternalResolutionMessage( externalResolutionMessage: $data['err_description'] ?? null)
    ->setDescription($data['description'])
;
```

Рисунок 4.9 – Перенесення отриманих даних на модель сервісу

Для цього існують два основні способи, це отримання сповіщення (callback) або відправити запит на отримання статусу. Існують і інші варіанти, наприклад відправити запит за якийсь період часу, та вже з того списку знайти свій платіж,

але такі випадки трапляються доволі рідко. Також важливо те, що сповіщення бувають відсутні, але це буває доволі не часто. Тож основним та присутнім у всіх платіжних системах способом перевірки стану платежу є опитування по API.

Слід зазначити, що не всі поля сервіс може підібрати самостійно, в деяких випадках необхідно, щоб користувач сам надав їх. Наприклад, якщо такі поля, як expired_date, action та order_id передає сервіс, то такі поля як return_url та language необхідно, щоб користувач передав сам, звісно, як варіант можна задати якісь данні за замовчуванням, але це вплине на конфігурабельність сервісу.

Варто розуміти, що якщо якогось параметру не буде враховано і буде спроба брати дані, яких не має, то сервіс поверне помилку, що не може їх обробити. Одним з важливих місць переносу даних є розуміння формату суми, в таблиці 4.3 можна побачити всі 4 варіанти, які можуть віддавати платіжні системи. Для цього реалізовано форматер ExtensionMoneyFormatter, куди необхідно передати суму, валюту та тип запису суми і він верне суму у форматі сервісу.

Таблиця 4.3 – Формат суми

Запис суми	Приклад
Строка з копійками	“0.01”
Строка з цілим числом	“1”
Число з копійками	0.01
Ціле число	1

Для того, щоб було зрозуміло наскільки важливо правильно реагувати на дані, які сервіс отримує від платіжної системи, далі буде розглянуто реальний приклад, як при одній невірній стрічці коду було втрачено велику кількість коштів.

Під час розробки системи онлайн платежів було реалізовано інтеграцію з однією платіжною системою (назва не буде наведена) і деякий час вона працювала коректно, але потім помітили, що до платіжної системи надходять

великі суми коштів, але до банку вони не доходять. Після такої знахідки було відключено PSP для того, щоб спочатку розібратися в ситуації, оскільки кожна хвилина її роботи приносила значні втрати.

Проблема була одразу у двох місцях:

– некоректна робота платіжної системи;

– неправильне реагування на відповіді від неї.

Отже, користувач (шахрай, який отримав кошти) створив платіж на сто тисяч гривень і система його перенаправляла на платіжну сторінку для того, щоб він заповнив свої реквізити та підтвердив платіж. Але після того, як він перейшов на цю сторінку він просто відкрив інспектор коду в браузері та змінив сто тисяч на одну гривню і підтвердив платіж. Приклад редагування зображено на рисунку 4.10.

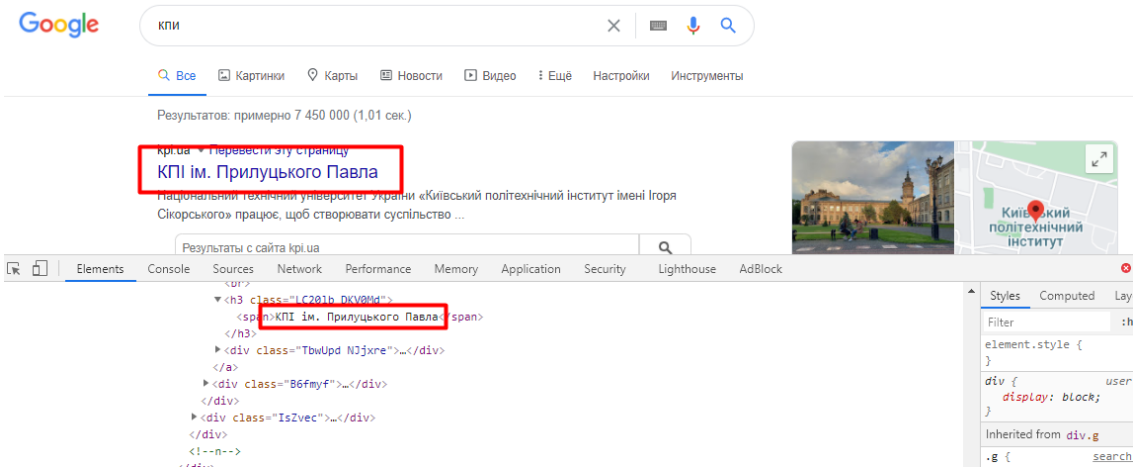

Рисунок 4.10 – Приклад редагування тексту в інспекторі браузера

Коли платіжна система обробила запит, вона надіслала оповіщення, що операція успішна і ця система онлайн платежів повідомила клієнтів, що операція на сто тисяч гривень успішна, але для того, щоб такого уникнути необхідно було просто подивитися, яку суму надіслали у цьому сповіщені.

Приклад того, як плативши лише одну гривню користувачу на гаманець надходили сотні тисяч гривень.

4.4 Маршрутизація

Одна з основних цінностей сервісу – управління великою кількістю підключених в систему аккаунтів. Важливим, а в більшості випадків навіть критичним аспектом такого управління, є маршрутизація платежів (виплат). Маршрутизація платежів визначає на який конкретний аккаунт буде прийнятий платіж в даному контексті (сума, валюта, карта, країна і т.д.). Маршрутизація безпосередньо впливає на два ключових для бізнесу значення: собівартість прийняття платежу і конверсія. Конверсія, в випадку, що стосується сервісу, це відношення кількості успішних операцій до загальної кількості операцій. Також за допомогою маршрутизації вирішуються і інші бізнес потреби, один з яких лімітування трафіку за певними аккаунтами (наприклад на аккаунт можна зараховувати платежі більше 15000 гривень або приймати платежі з іноземних карт).

Через актуальності і функціональності цього інструменту, до нього висувається ряд вимог: простота розуміння і управління, прозорість роботи, аудит змін, можливість аналізу результату роботи.

У додатку Д можна побачити діаграму послідовності створення платежу у сервісі.

Слід зазначити, що User (тобто користувач), це не обов’язково людина, бо сервіс інтегрується з сайтами клієнтів і якісь дані може передавати сайт чи система клієнтів. На рисунку 4.11 можна побачити приклад запиту на створення платежу. Для цього необхідно передати суму, валюту, ідентифікатор платежу в системі клієнта та інші необов’язкові поля.

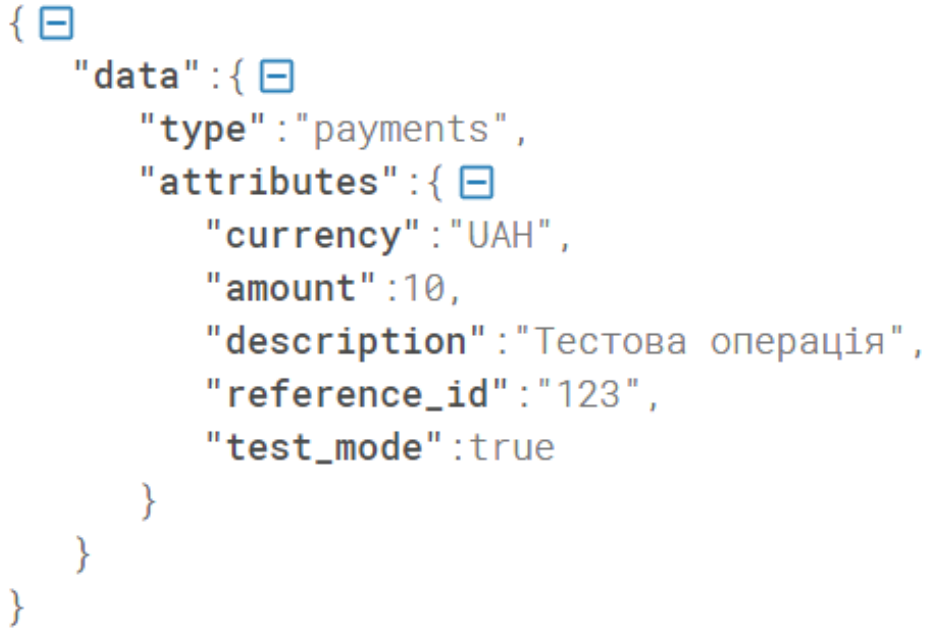

Рисунок 4.11 – Запит на створення платежу

На основі цих полів система розуміє, які маршрути необхідно передати на RDP. RDP – місце, де відбувається вирішення по якому платіжному маршруту піде платіж. Звісно, сервіс, який представлений в магістерській дисертації не володіє штучним інтелектом, то ж клієнти, які інтегрувались до нас повинні правильно налаштувати правила маршрутизації, але на майбутнє плануються можливість за допомогою метрик сервісу впливати на якісь процеси, наприклад якщо по одному маршруту отримаємо лише неуспішні операції то він автоматично виключиться.

В якості базової моделі була обрана структура умовного оператора (if-then-else), така структура є природною і зрозумілою навіть не для IT фахівця. Модель схеми маршрутизації має деревоподібну рекурсивну структуру (граф), в вершинах якої знаходяться або умовний оператор або стратегія. Умова визначає наступне ребро при обході графа. Кінцеві вершини завжди представлені стратегіями. Відповідно і результатом обходу графа є певна стратегія. На рисунку 4.12 можна побачити приклад алгоритму маршрутизації.

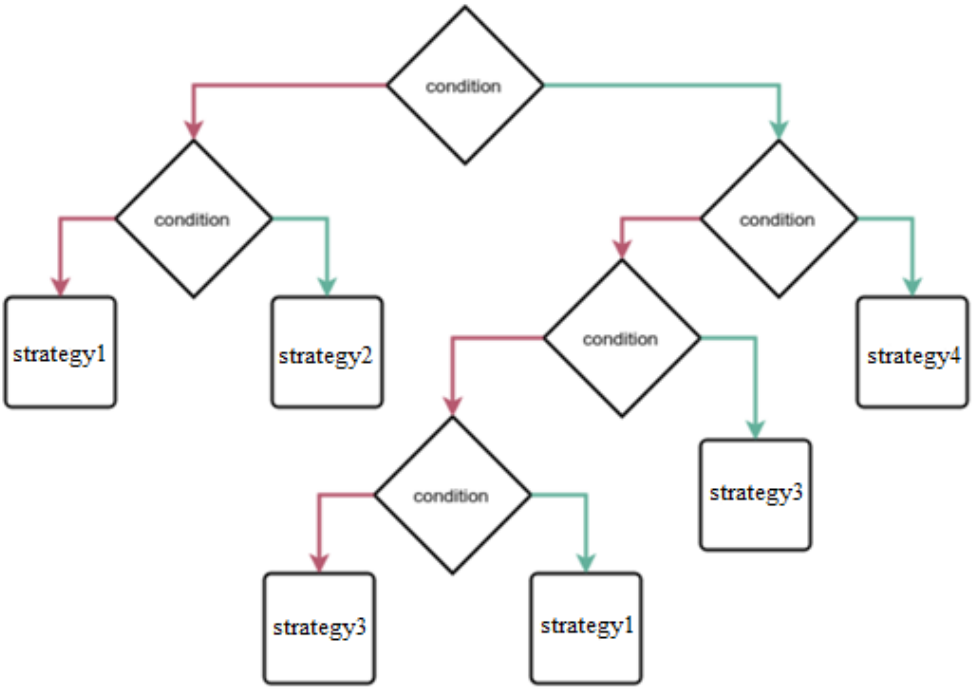

Рисунок 4.12 – Приклад алгоритму маршрутизації

Умов може буде скільки завгодно, наприклад сума повинна бути більше якогось значення або електронній адрес користувача можна вказати. Для розуміння простий приклад: підключено два аккаунти, для одного вигідніше проводити платежі менше 100 гривень, а для іншого більше. Тому необхідно лише вказати, що платежі менше 100 гривень йдуть на маршрут першого аккаунта, а більше 100 – на другий.

Стратегій є всього 5:

– Direct – вказується маршрут, на який весь час будуть йти платежі;

– Optimal – маршрут буде вибиратися випадково;

– Weight – можна ставити відсотки, на який маршрут скільки трафіку пускати. Слід зазначити, що якщо двом маршрутам поставити по 50% це не означає, що із десяти платежів п’ять піде на перший, а п’ять на інший, оскільки використовується ймовірність і вибір маршруту є приблизним.

– By fee – вибирає маршрути з найменшою комісією;

– Traversal – після того як вже був трафік по маршрутам, система розуміє по яким з них найкраща конверсія і вибиратиме саме їх, але дана стратегія ще не реалізована.

Також важливою частиною маршрутизації є використання курсу валют, всі курси тягнуться з одного ресурсу та зберігаються в Redis для високої швидкості зчитування. Redis – розподілене сховище пар ключ-значення, вона зберігає усю інформацію у оперативній пам'яті, але обіцяє зберігання цих даних на довгий проміжок часу. Максимальна продуктивність даного рішення сто тисяч запитів в секунду. Спираючись на ці дані можна сміливо говорити, що дана система повністю задовольняє необхідності сервісу.

На рисунку 4.13 можна побачити метод, який викликається для конкретної пари, та відає курс валют для неї. Як бачимо, якщо передали однакові валюти, то просто повертається одиниця, щоб зайвий раз не йти до бази. Хоч і можна сказати, що курси лежать у кеші, але за допомогою команди «bin/console rate-scheme:rates:sync», яка запускається через кожний проміжок часу вони оновлюються. Цю команду необхідно запускати доволі часто, оскільки національні валюти можуть часто змінюватися при якихось кризах, а навіть коли їх немає, то курс віртуальної валюти може змінюватися кожну хвилину.

```php
public function getRateByScheme(RateScheme $scheme, string $pair): float
{
    $currencyPair = CurrencyPair::fromPair($pair);

    /** @var Currency $baseCurrency */
    $baseCurrency = $this->currencyRepository->find($currencyPair->getBase());

    /** @var Currency $quoteCurrency */
    $quoteCurrency = $this->currencyRepository->find($currencyPair->getQuote());

    $base = null !== $baseCurrency->getParent() ? $baseCurrency->getParent()->getCode() : $baseCurrency->getCode();
    $quote = null !== $quoteCurrency->getParent() ? $quoteCurrency->getParent()->getCode() : $quoteCurrency->getCode();

    if ($currencyPair->isSameCurrencies() || $baseCurrency->isSameParent($quoteCurrency)) {
        return 1;
    }

    return $this->liveRateRepository->getRate($scheme, (new CurrencyPair($base, $quote))->getPair());
}
```

Рисунок 4.13 – Пошук курсу для пари валют

4.5 P2P-платежі

P2P-платежі – перекази с карти на карту між фізичними особами з карти на карту. Це тип платежів коли гроші не повністю зараховується на рахунок банку, а містяться на віртуальному рахунку. Для клієнтів це вигідно, тому що при звичайній взаємодії клієнт платить комісію під час прийому платежу, а потім під час виплати, але при P2P-платежі комісія знімається лише один раз, тому даний вид операції є пріоритетним для клієнтів.

Для таки платежів було створено окремо сутність P2pPayment, де міститься вся необхідна інформація для виплати. Ця сутність створюється тоді і тільки тоді, як платіж отримав статус успіх по маршрутах в яких стоїть позначка, що він P2P. На рисунку 4.14 можна побачити структуру P2pPayment.

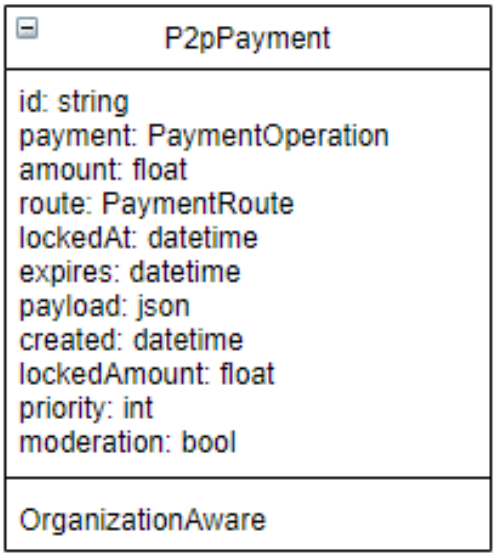

Рисунок 4.14 – Структура класу P2pPayment

Його поля:

– id – унікальний ідентифікатор;
– payment – посилання на платіж;
– amount – сума P2P-платежу;
– route – посилання на платіжний маршрут;

– lockedAt – час, в який P2P-платіж взято в обробку;

– expires – дата, коли платіж видаляється з платіжної системи;

– payload – місце де зберігається уся додаткова інформація, що необхідна під час створення виплати;

– created – дата створення платежу;

– lockedAmount – сума, яка була взята в обробку;

– priority – впливає на пріоритетність вибору P2P- платежу для створення виплати;

– moderation – флаг, означає що з платежем щось не так і виплачуватися він не буде.

Після того, коли P2P-платіж став успішним і на повну суму, то він видаляється, це чи не єдина сутність, яка видаляється з системи та потім ніде (за винятком логів) не фігурує. Якщо виплата була на суму менше платежу, то вона просто віднімається від загальної суми. І у випадку коли виплата стала неуспішною, то з P2P-платежу знімається lockedAt і його можна використовувати далі. Також, якщо після того, як сервіс спробував створити виплату отримали помилку, що такого платежу не існує, то окрім його розблокування ставим його на модерацію (moderation = true) і вже після цього клієнту необхідно власноруч з ним розібратися. Ще існує можливість власноруч поставити платіж на модерації.

На рисунку 4.15 можна побачити вибірку P2P-платежів починаючи з найстаріших. Найстаріші це ті, які вже знаходяться в системі більше половини строку існування, тобто якщо строк життя 30 днів, то вже через 15 днів він буде більш пріоритетним.

```php
public function findByExpirationDate(
    PaymentRoute $route,
    string $amount,
    AmountLimit $limits,
    bool $splitMode = false,
    float $days = 14.0
): array {
    $qb = $this->createQueryBuilder('p2p');

    $expirationDate = Carbon::now()->addHours($days * 24);

    $qb
        ->andWhere('p2p.paymentRoute = :route')
        ->andWhere('p2p.lockedAt is null')
        ->andWhere('p2p.moderationRequired = false')
        ->andWhere('p2p.amount >= :amountLimitMin')
        ->andWhere('p2p.amount <= :amountLimitMax')
        ->andWhere('p2p.expires < :expirationDate')
        ->setParameter('route', $route)
        ->setParameter('expirationDate', $expirationDate)
        ->setParameter('amountLimitMin', $limits->getMin())
        ->setParameter('amountLimitMax', $limits->getMax())
        ->orderBy('p2p.expires', 'ASC')
        ->setMaxResults(100);
```

Рисунок 4.15 –Вибірка P2P-платежів

Також для того, щоб виплатити платіж необхідно, щоб він вже не був взятий в обробку іншим процесом, тобто параметр lockedAt пустий та не був на модерації. Ну і як для звичайної виплати P2P-платіж повинен задовільнити лімітам маршруту. Якщо по таким критеріям нічого не буде знайдено, то будемо шукати P2P-платіж просто по сумі.

4.6 Робота з організацією

На додатку Е можна побачити діаграму класів організації. На перший погляд може здатися, що організація ніяк не пов'язана з іншими сутностями, але вони просто реалізовують інтерфейс OrganizationAwareInterface в методі якого

getOrganization() необхідно повернути організацію. Більшість полів організації містять лише інформаційний характер.

Серед інших сутностей слід виділити OrganizationApiKey, тобто ключ доступу до організації, оскільки список довірених ip адрес було винесено до домішки (trait) та створений окремий інтерфейс, для того, щоб в майбутньому була можливість застосувати його в інших сутностях, які будуть цього потребувати. Trait – механізм мови PHP, який з'явився у версії 5.4, основна його ціль це заміна множинного успадкування, оскільки дана мова не має такої можливості. На рисунку 4.16 можна побачити використання trait у класі OrganizationApiKey.

```php
/**
 * Class OrganizationApiKey.
 */
class OrganizationApiKey implements
    IpWhiteListInterface,
    OrganizationAwareInterface
{
    use IpWhiteListTrait;
```

Рисунок 4.16 – Використання trait

MemberRole – це ролі організації, а OrganizationMember – її учасники. Більшість полів стандартна і зрозуміла, але на деякі слід звернути увагу детальніше:

– creator – ідентифікатор учасника, який створив цю роль або запросив учасника;

– accessScope – зберігає назви частин сервісу, до якої доступ дозволений.

Опис таблиць у базі даних:

У таблиці 4.4 наведено опис таблиці organizations

Таблиця 4.4 – Таблиця organizations

Назва колонки	Тип даних	Призначення
id	varchar(64)	Унікальний ідентифікатор, який надається для кожній організації
email	varchar(64)	Електронна адреса
domainName	varchar(64)	Доменне ім’я
phone	varchar(64)	Телефон
site	varchar(64)	Посилання на сайт
features	json	Особливі можливості, які присутні у організації
supportUrl	varchar(64)	Посилання на службу підтримки, якщо є якісь труднощі
liveSecret	varchar(300)	Ключ для шифрування сповіщень з реальними операціями
testSecret	varchar(300)	Ключ для шифрування сповіщень з тестовими операціями
roles	json	Ролі
pspDirectoryCode	varchar(64)	Якщо організація являється платіжною системою, то можна вказати її код
encryptionKeys	json	Ключі шифрування, якщо є в цьому необхідність
created	datetime(3)	Дата створення

Назва колонки	Тип даних	Призначення
contactEmail	varchar(64)	Електронна адреса для зв’язку з підтримкою
address	varchar(256)	Адреса
hasLogo	bool	Чи має організація логотип
webhookVersion	varchar(16)	Версія сповіщення

У таблиці 4.5 наведено опис таблиці members

Таблиця 4.5 – Таблиця members

Назва колонки	Тип даних	Призначення
id	varchar(64)	Унікальний ідентифікатор члена організації
inviteEmail	varchar(64)	Електронна адреса на яку було вислано запрошення приєднатися до організації
memberRole	varchar(64)	Посилання на MemberRole
status	varchar(64)	Статус користувача, спочатку він запрошений, потім активний, а якщо він вже не працює в організації то можна його архівувати
name	varchar(64)	Ім’я
options	json	Особливі можливості, які присутні у члена організації, наприклад не показувати тестові операції
organization	varchar(64)	Посилання на організацію

Назва колонки	Тип даних	Призначення
accessScope	json	Доступні частини сервісу, до яких має доступ користувач
individualAccessScope	bool	Чи використовувати доступи ролі користувача або дати йому особливі
inviteToken	varchar(64)	Ключ для реєстрації
description	text	Опис
joined	datetime(3)	Дата коли користувач приєднався
created	datetime(3)	Дата створення
creator	varchar(64)	Посилання на користувача, який запросив даного
user	varchar(64)	Посилання на користувача сервісу

У таблиці 4.6 наведено опис таблиці member_roles

Таблиця 4.6 – Таблиця member_roles

Назва колонки	Тип даних	Призначення
id	varchar(64)	Унікальний ідентифікатор ролі
creator	varchar(64)	Посилання на користувача, який створив роль
code	varchar(64)	Код ролі
created	datetime(3)	Дата створення
accessScope	json	Доступні частини сервісу, до яких має доступ роль

Назва колонки	Тип даних	Призначення
name	varchar(64)	Назва ролі
organization	varchar(64)	Посилання на організацію
description	text	Опис
sortWeight	int	Показує наскільки важлива роль при порівнянні з іншими

У таблиці 4.7 наведено опис таблиці org_api_keys

Таблиця 4.7 – Таблиця org_api_keys

Назва колонки	Тип даних	Призначення
id	varchar(64)	Унікальний ідентифікатор ключа
ipWhiteList	json	Список ip адрес з яких можна робити запити до сервісу
permissions	json	Доступні частини сервісу
individualAccessScope	bool	Чи використовувати всі доступи або дати йому особливий набір

Висновки до розділу 4

У даному розділі було описано структуру коду та архітектуру системи. Розглянули основні принципи взаємодії з платіжними системами та способи взаємодії з ними. Також прояснено найменування сервісів та як оброблювати різні статуси від платіжних систем. Було детально описано процес проходження платежу. Роз’яснено роботу усіх стратегій та використання умов для налаштування маршрутизації. Описано принци роботи та вибірку P2P-платежів використовуючи схеми та код. Також описано діаграму класів організації.

5 ІНСТРУКЦІЯ КОРИСТУВАЧА

5.1 Підключення та взаємодія з аккаунтом

Однією з перших взаємодій з сервісом є підключення будь-якої платіжної системи. Розглянемо на прикладі LiqPay. Під час вибору платіжної системи можна подивитися, які властивості вона має, як на рисунку 5.1.

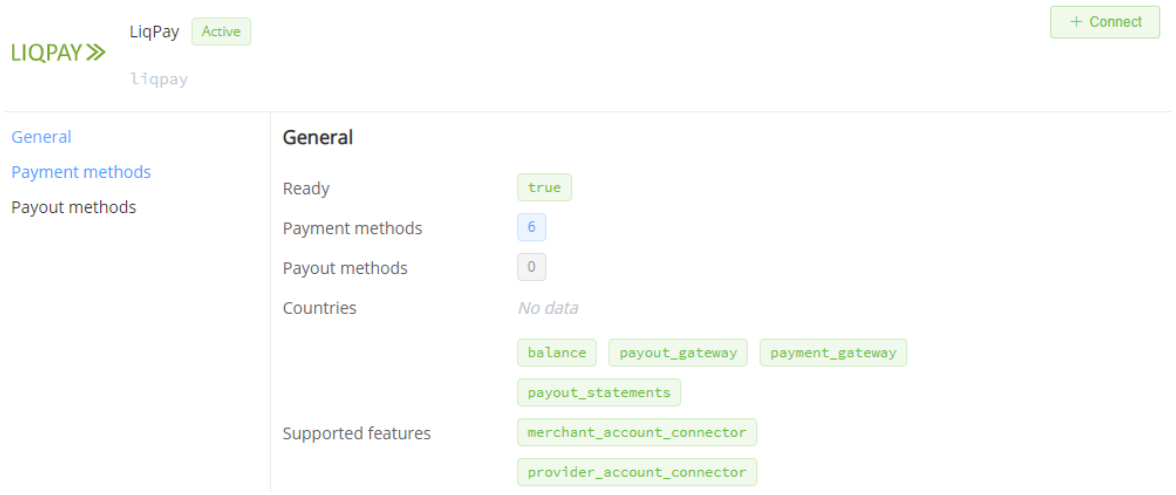

Рисунок 5.1 – Доступні можливості платіжної системи

На прикладі можна побачити, що у даної платіжної системи є можливості оновлення балансу, прийом/виплата платежів та інше. На ці дані зручно спиратися, коли шукаєте нові платіжні системи для інтеграції. Наприклад, для клієнтів необхідно кожен раз перевіряти доступний для них баланс, але платіжна система не віддає таких даних по API, тому можна одразу відкинути дану PSP зі списку потенційних. Також можна дізнатися, які платіжні методи доступні.

Серед усіх платіжних систем, необхідно вибрати, яка задовольняє усім критеріям і потім натиснути на кнопку «Підключити», як зображено на рисунку 5.1. На рисунку 5.2 можна побачити приклад вікна, яке відкрилося, та побачити список необхідних полів.

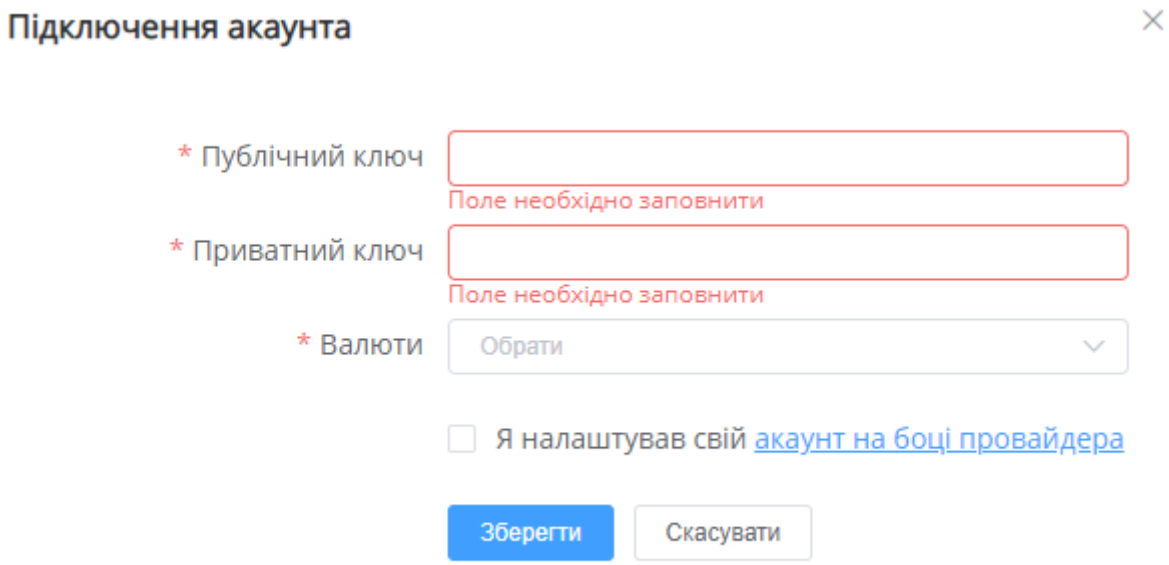

Рисунок 5.2 – Підключення LiqPay

Перші два поля необхідно взяти з особистого кабінету, як показана на рисунку 5.2. Поле валюта необхідно для того, щоб сервіс розумів під яку валюту створювати аккаунти та маршрути. Також необхідно звернути увагу на те, що необхідно поставити галочку, що ви налаштували аккаунт, вона просто діє як нагадування і ніяк не перевіряє чи ви аккаунт дійсно правильно налаштувати. На рисунку 5.3 можна побачити відмічені поля, які потрібно увімкнути та заповнити, щоб сервіс працював коректно. В інших платіжних системах бувають випадки коли необхідно зв’язатися з менеджером для того щоб він сам увімкнув необхідні опції. Деякі з платіжних систем не мають додаткових налаштувань, але необхідно бути в цьому впевненим.

Після того, як клієнт ввів всі необхідні поля на рисунку 5.2 та обрав в якій валюті необхідно створити аккаунт та відповідні йому маршрути, сервіс створить їх.

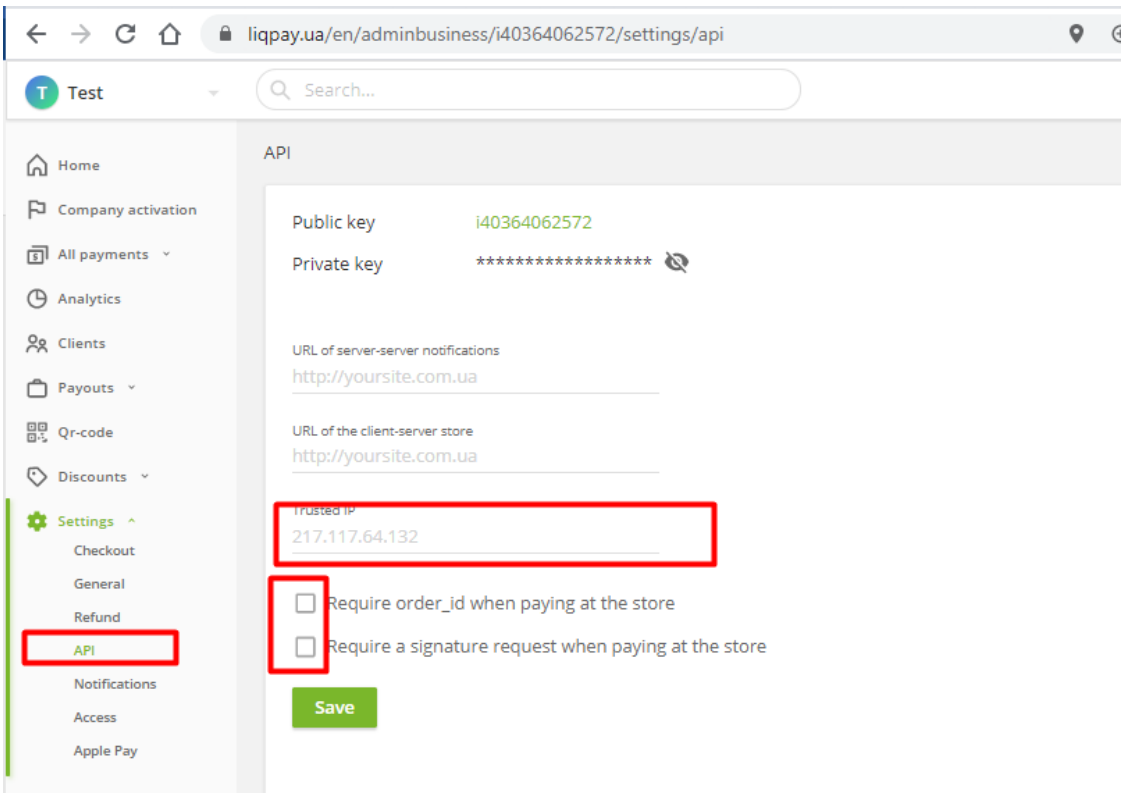

Рисунок 5.3 – Налаштування LiqPay

Слід зазначити, що це відбудеться при коректності даних, в іншому випадку повернеться помилку підключення, приклад такої помилки можна побачити на рисунку 5.4.

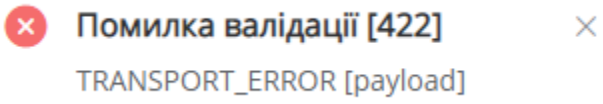

Рисунок 5.4 – Модальне вікно з помилкою.

Якщо з’явилася помилка, то це означає, що треба перевірити ключі доступу до системи, необхідні опції або зв’язатися з представниками сервісу чи платіжної системи для більш детального пояснення.

Якщо всі доступи вірні, то перед клієнтом одразу відкриється вікно з аккаунтом та маршрутами, як це показано на рисунку 5.5.

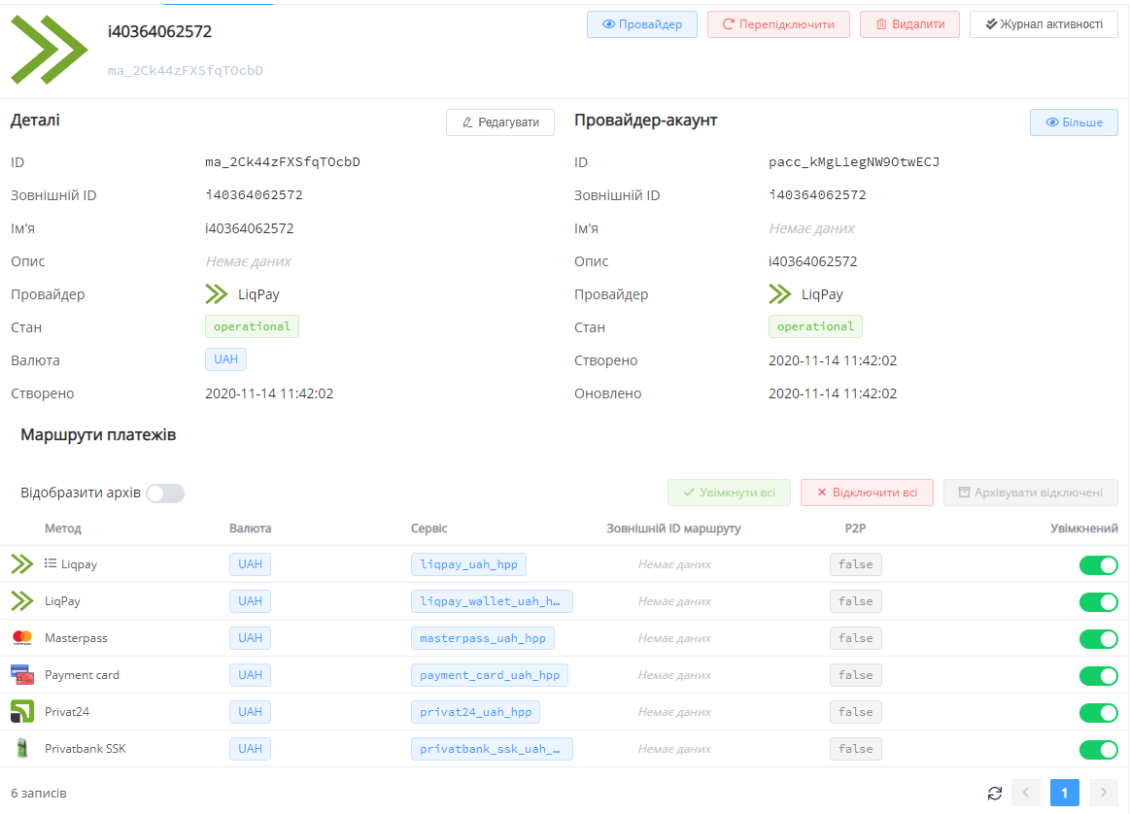

Рисунок 5.5 – Вікно підключеного аккаунту

На цьому вікні можна побачити основну інформацію про аккаунт та за необхідності вимикати чи вмикати маршрути. Якщо ви вимкнули маршрути і розумієте, що вам вони вже не знадобляться, то можна їх архівувати, щоб вони вам не заважали. Звісно в будь-який момент їх можна дістати з архіву.

Також можна побачити, що у кожного конектора є поле стан, воно відображає чи можна зараз користуватися конектором. На рисунку 5.5 можна бачити, що зараз стан – operational, тобто аккаунт готовий до роботи. Також є статус outdated_credentials, він з’являється коли сервіс отримав відповідь, що ключі доступу неправильні та connection_error коли неможливо здійснити запит, наприклад, коли платіжна система перестала працювати чи довго відповідає. Дана функція ще не є реалізованою, оскільки ще є питання до доцільності її впровадження.

На рисунку 5.6 продемонстровано опції редагування аккаунту для виплат. Окрім можливості змінити ім’я та додати опис він також має три важливі функції, які будуть описані далі.

Мінімально допустимий баланс це межа, нижче якої не може опускатися баланс аккаунту. Наприклад, якщо границя встановлена в тисячу гривень, а у вас на балансі дев’ятсот, то сервіс не дасть зробити виплату більш ніж на 100 гривень. Якщо з технічної сторони все зрозуміло, то зі сторони фінансів це важливий інструмент для регулювання, оскільки банки або платіжні системи можуть висувати такі умови.

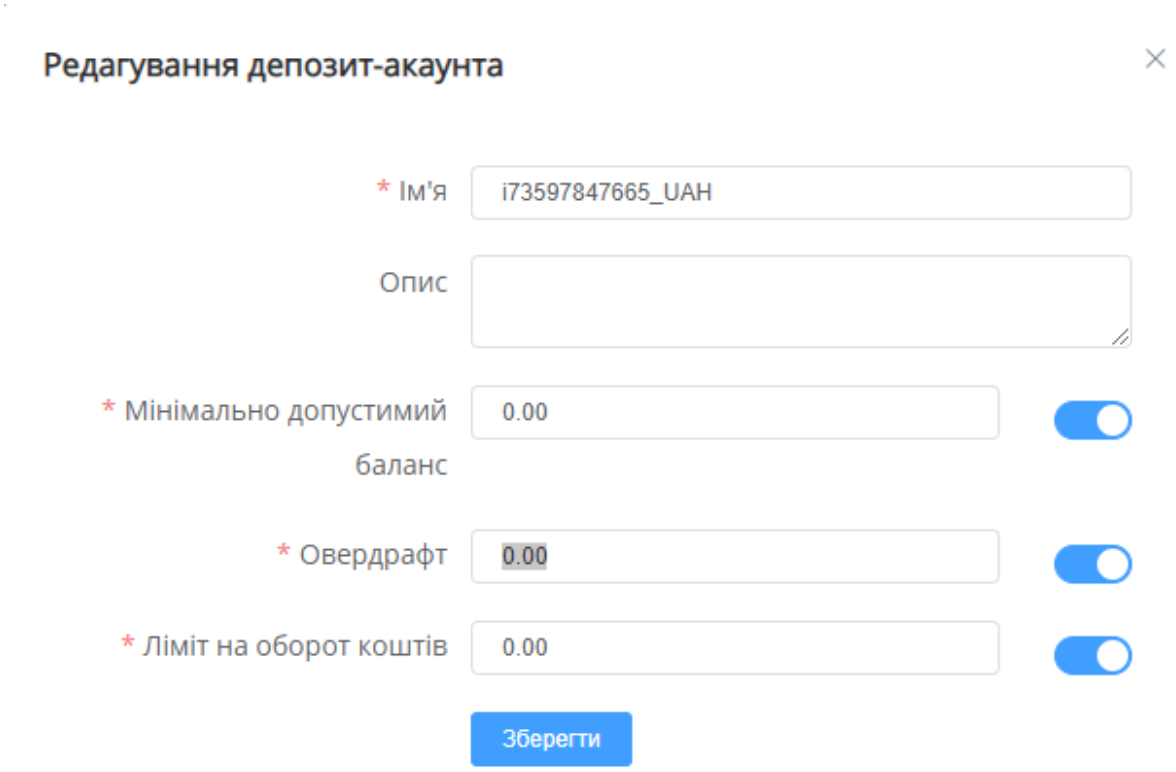

Рисунок 5.6 – Редагування аккаунту для виплат

Наступним важливим елементом є овердрафт, він також є важливим фінансовим інструментом. Овердрафтом називаються кредитні кошти, які надаються банком, коли клієнту не вистачає власних коштів. Загалом овердрафт це короткостроковий кредит з деякими перевагами над звичайним, але не будемо вдаватися в деталі, оскільки це чисто фінансова сторона. Сервіс під час роботи з овердрафтом просто додаю його до звичайного балансу, оскільки немає необхідності оперувати ним, як двома різними сумами. А клієнту зручно бачити розділення на власні кошти та кредитні.

Також було додано функцію на ліміти коштів за день. Наприклад, якщо ви встановили ліміт у тисячу гривень і вже вичерпали його, то наступну виплату ви зможете зробити на наступний день або якщо вимкнуте цю функцію. Ліміт на кошти також є важливим фінансовим інструментом і допоможе у урегулюванні виплат.

Платіжні системи доволі часто додають новий функціонал та додають нові способи оплати чи виплат, тому додано кнопку на перепідключеня аккаунту, щоб ці платіжні методи з’явилися в сервісі. Також не слід забувати, що не можна просто так міняти ключі доступу у платіжній системі не змінивши їх одразу у нас, тому що сервіс може одразу перестати працювати. Якщо у вас все таки виникла необхідність змінити ключі, то можна перепідключити аккаунт з потрібними вам даними, як це показано на рисунку 5.7. На цьому етапі це все що необхідно знати про підключення та взаємодію з аккаунтом.

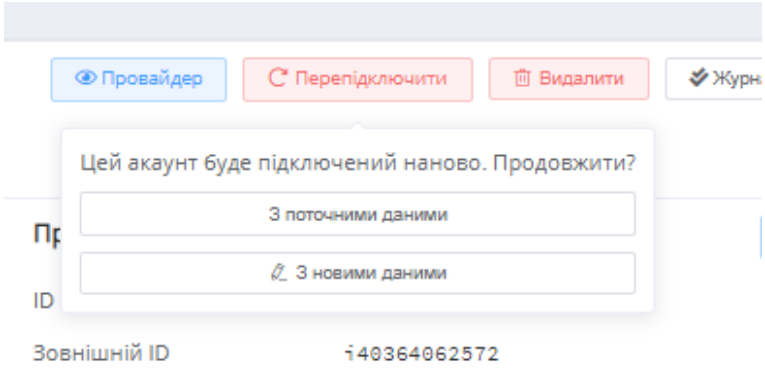

Рисунок 5.7 – Перепідключення аккаунту

Однією з переваг даного сервісу є те, що в ньому присутній журнал активності в якому можна побачити хто та коли змінював щось у сутностях системи. Це дуже важлива функція коли у клієнта сервісу працює більше однієї людини, тому що хтось можете нашкодити системі змінивши якісь правила чи вимкнувши маршрут. Також це допомагає полагодити щось просто повернувши все назад, що було зроблено.

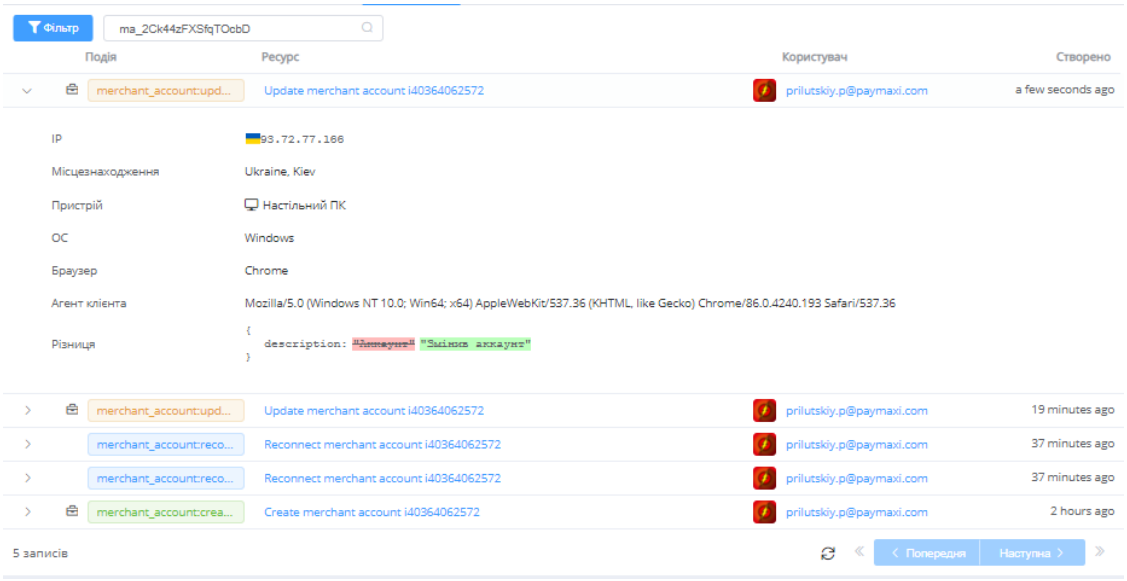

Рисунок 5.8 – Журнал активності

На рисунку 5.8 можна побачити журнал активності аккаунту, який нещодавно підключили. Перший запис найсвіжіший, останній – самий старий. Можна одразу побачити, що користувач prilutskiy.p@paymaxi.com дві години назад створив MerchantAccount та можна також побачити його ідентифікатор. Потім цей же користувач два рази перепідключив аккаунт та два рази змінив дані в ньому. В останньому записі видно, що можна побачити IP користувача, місцезнаходження та інші його параметри. Найцікавішим є те, що відображено дані, які були змінені, раніше опис був «Акканут», а тепер – «Змінив аккаунт».

Видалити аккаунт можна тільки за умови, що по ньому ще не було створено платежів, оскільки платежі пов’язані з маршрутами, а ті в свою чергу з аккаунтами, тобто платіж без аккаунту існувати не може.

Взаємодія з маршрутами трохи нагадує взаємодію з аккаунтами, там також можна подивитися основну інформацію по маршрутам і також має журнал активностей. Основною задачею маршрутів є регулювання лімітів та комісії. На рисунку 5.9 можна побачити приклад редагування маршруту.

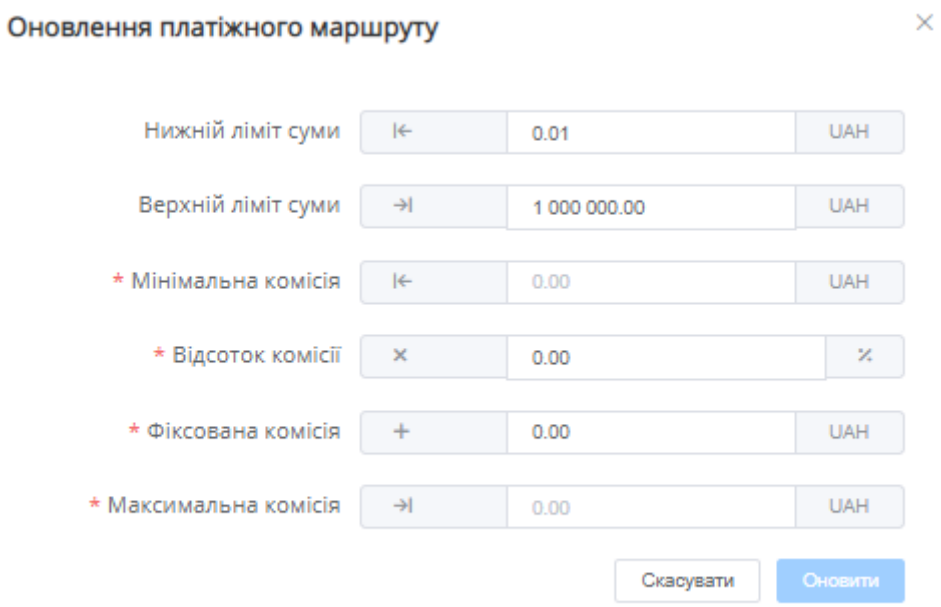

Рисунок 5.9 – Налаштування комісій

Перші два поля для того, щоб регулювати суму, яка буде йти по такому платіжному методу, якщо сума не буде підходити сервіс просто відкине цей маршрут і візьме тільки ті, що підходять. Другою можливістю і однією з найважливіших цінностей є налаштування комісії, бо це саме те місце через яке клієнти можуть заробляти гроші. Підрахунки дуже прості, якщо ви заповнюєте суму фіксованої комісії то вона додається до суми платежу, якщо відсоток заповнюєте, то відсоток від платежу додається. Коли комісія вийшла менше мінімальної то мінімальна комісія становиться комісією платежу, а якщо вище максимальної то максимальна становиться.

5.2 Робота з платежем

На рисунку 5.10 продемонстровано вікно успішного платежу. Тут можна знайти всю необхідну інформацію від ідентифікатора платежу до часу його обробки. Справа можна побачити хронологію подій, тобто від часу створення до часу обробки. Звідси можна побачити по якому маршруту та аккаунту був створений платіж.

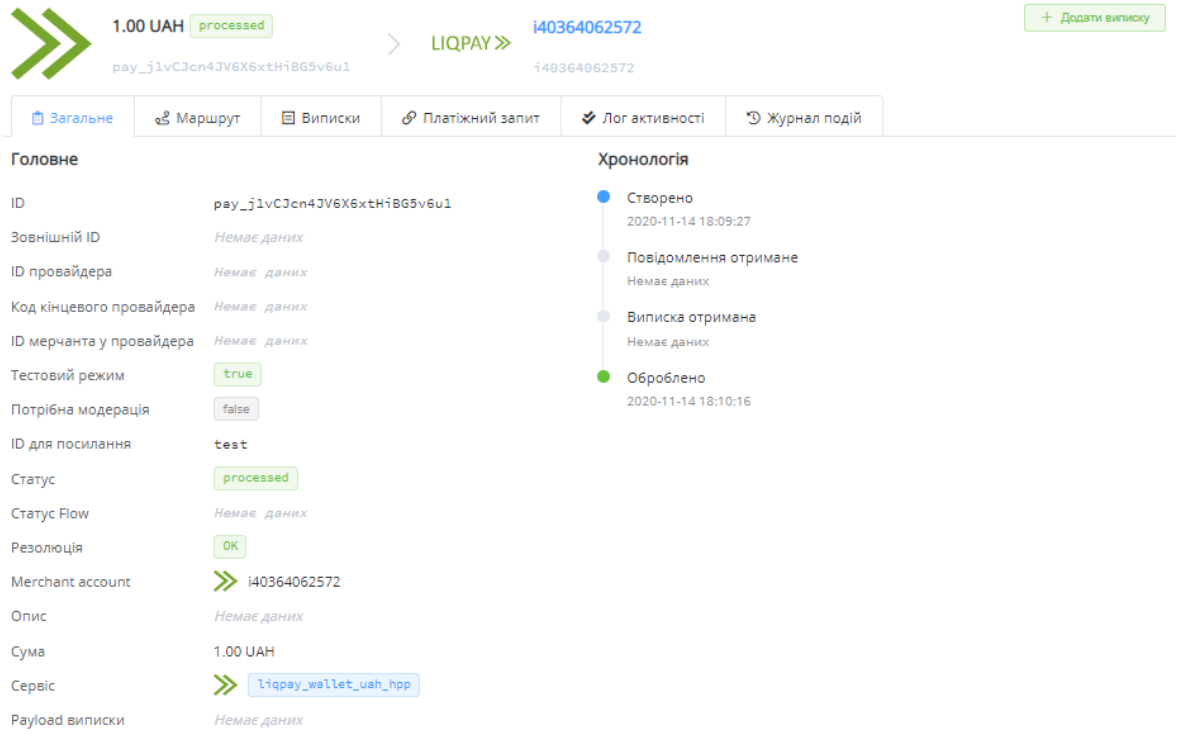

Рисунок 5.10 – Вікно платежу

Наступне, що можна побачити це посилання на виписки від платіжної системи. На рисунку 5.11 можна побачити приклад спочатку успішної виписки, а потім неуспішної. Всього в сервісі існує три типи виписок:

– Notification – тобто callback, це коли платіжна система сама сповістить сервіс про зміну статусу;

– Reconciliation – коли сервіс сам пішов на PSP, щоб запросити статус;

– Manual – ручна виписка.

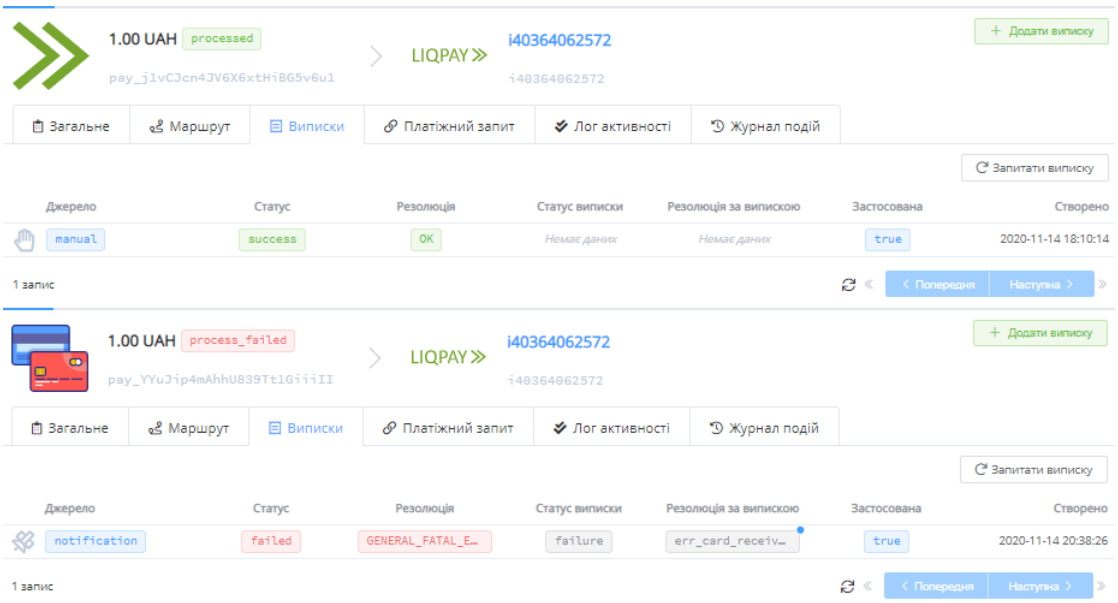

Рисунок 5.11 – Виписки

Ручна виписка необхідна для того, щоб закривати платежі, які за якихось умов не можуть закритися автоматично. Наприклад, якщо отримали якійсь новий і невідомий статус, то клієнту необхідно дізнатися у платіжній системі в якому статусі знаходиться платіж і після цього фіналізувати його. На рисунку 5.12 можна побачити приклад як можна зафіналити платіж, також при фіналізації можна вказати додаткові данні.

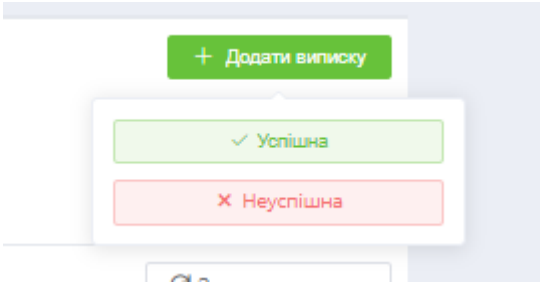

Рисунок 5.12 – Додавання виписки

5.3 Маршрутизація

Налаштування маршрутизації починається з переліку усіх сервісів як це показано на рисунку 5.13.

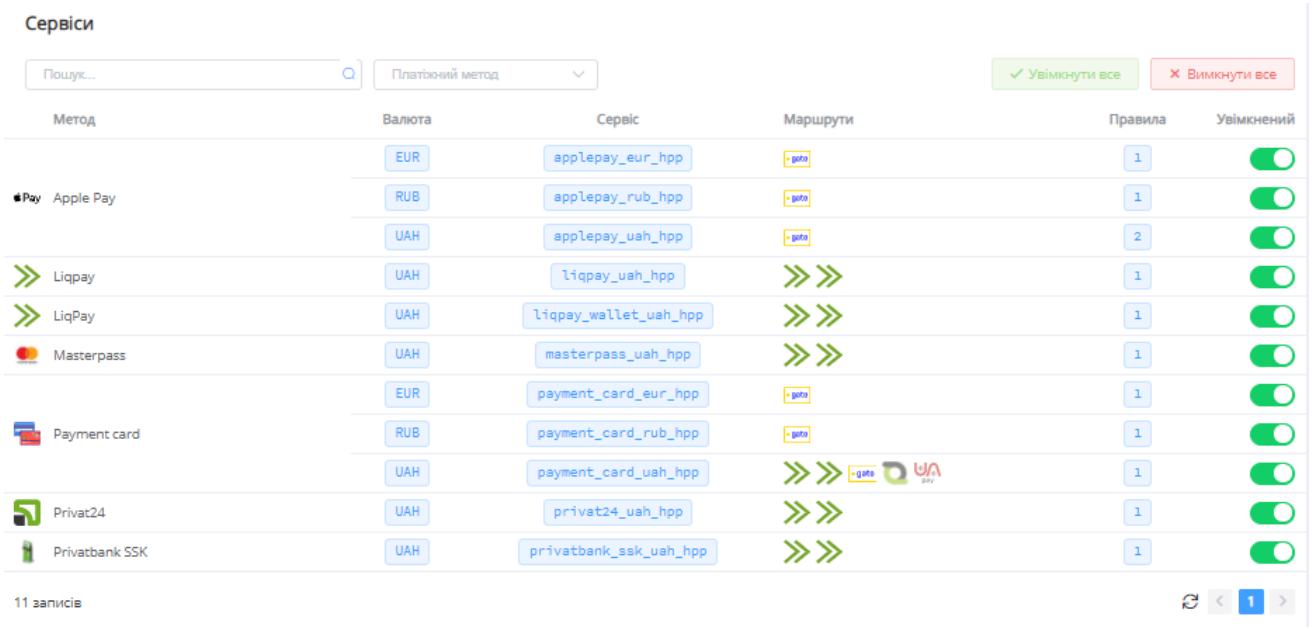

Рисунок 5.13 – Платіжні методи

Сервісом, як було вже описано раніше, є платіжний метод під конкретну валюту та флоу, тобто ви можете підключити хоч сто різних платіжних систем, а сервіс буде все рівно один, наприклад payment_card_uah_hpp. Або навпаки, навіть у одної платіжної системи може бути декілька сервісів. Сама маршрутизація відбувається в рамках одного сервісу, це потрібно розуміти, щоб правильно її налаштовувати.

На рисунку 5.14 можна побачити приклад правил сервісу.

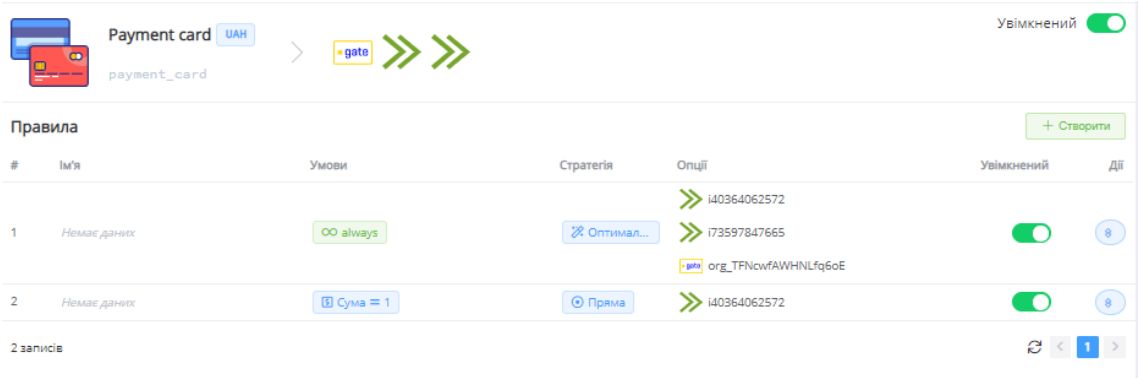

Рисунок 5.14 – Правила сервісу

Правил можна створювати скільки завгодно, але слід вважати на складність підтримки такої маршрутизації. Кожне правило налаштовується окремо і кожне

з них можна вимкнути. Також правило можна видалити зовсім, але треба бути в цьому впевненим, оскільки цю дії відмінити вже не можна буде. Варто сказати, що правила виконуються в тому порядку, як вони відображені. Порядок можна змінювати піднімаючи якісь правила вище, а інші навпаки опускати.

На рисунку 5.15 зображено налаштування правила та вибір його умов та стратегій. Можна бачити, що додано правило, що сума для цього правила потрібна бути рівної одиниці, а стратегія обрана по вагам.

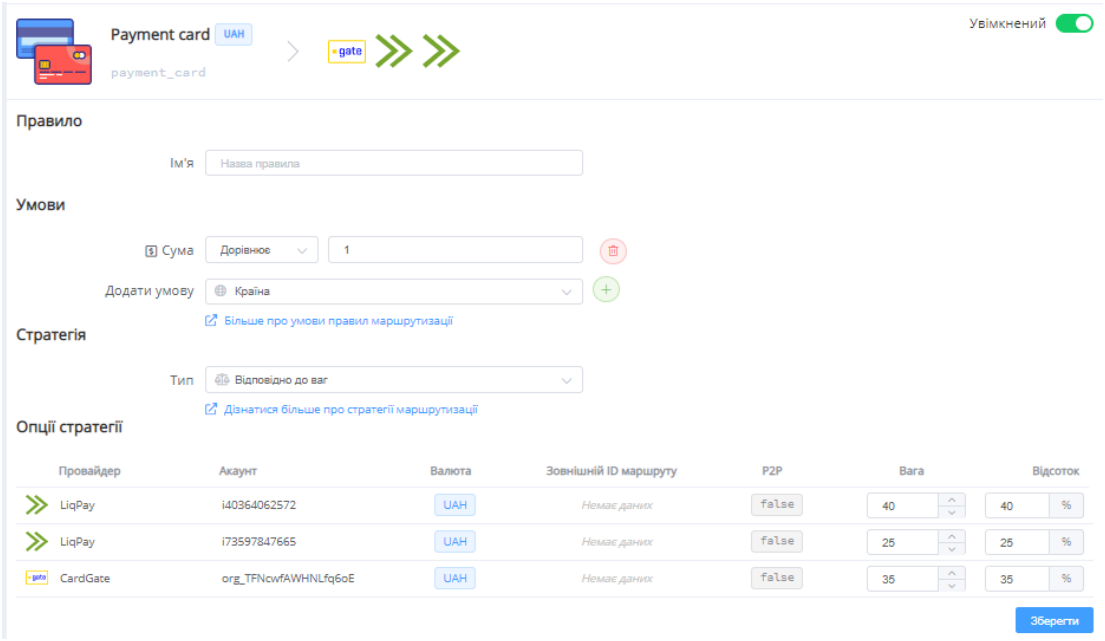

Рисунок 5.15 – Налаштування правила

У правила краще заповнювати ім’я, щоб потім легко орієнтуватися у них, але це не обов’язково. Після цього необхідно вибрати умови, їх може бути скільки завгодно, ось приклад кількох з них:

– країна з якої йде платіж;
– email користувача;
– сума;
– день тижня;
– додатково передане поле;

За бажанням умови можуть додаватися, але ті що є вже покривають більшість забаганок клієнтів. І тільки в тому разі, коли всі перевірки пройдені коректно вступає в силу стратегію, яку було обрано. Якщо якась з перевірок не пройшла, то йде перехід до наступного правила, а якщо і там не пройде умови і правила закінчяться то буде помилка, що немає доступних маршрутів і платіж буде неуспішний.

У додатку Ж можна побачити діаграму вибору стратегії. Спочатку береться перше правило і перевіряється його перша умова. Якщо перевірка умови проходить успішно, то перевіряється наступна доки всі не закінчяться. В тому випадку, колі всі перевірки пройшли вдало то використовуємо стратегію, яка зазначена у правилі. Якщо якась з умов не підійшла, то переходимо до наступного правила, у тому випадку, коли умови залишилися, то перевірка правил відбувається за алгоритмом описаним вище. А коли правил не залишилось то повертається результат про те, шо не знайдено підходящого маршруту.

Наступним і мабуть останнім важливим інструментом у маршрутизації є можливість налаштування курсів. Це є значною перевагою оскільки при стрибках валюти можна вказати курс за допомогою комісій трохи більше для того, щоб не втратити коштів. На рисунку 5.16 зображено редагування курсу гривень в долари.

Можна ставити як відсоток комісії так і фіксовану і одразу можна побачити який курс буде у користувача.

Важливе зауваження, що курс валют, наприклад, з гривень у долари та з доларів у гривні це два зовсім різних курси, тому і налаштовувати їх необхідно окремо.

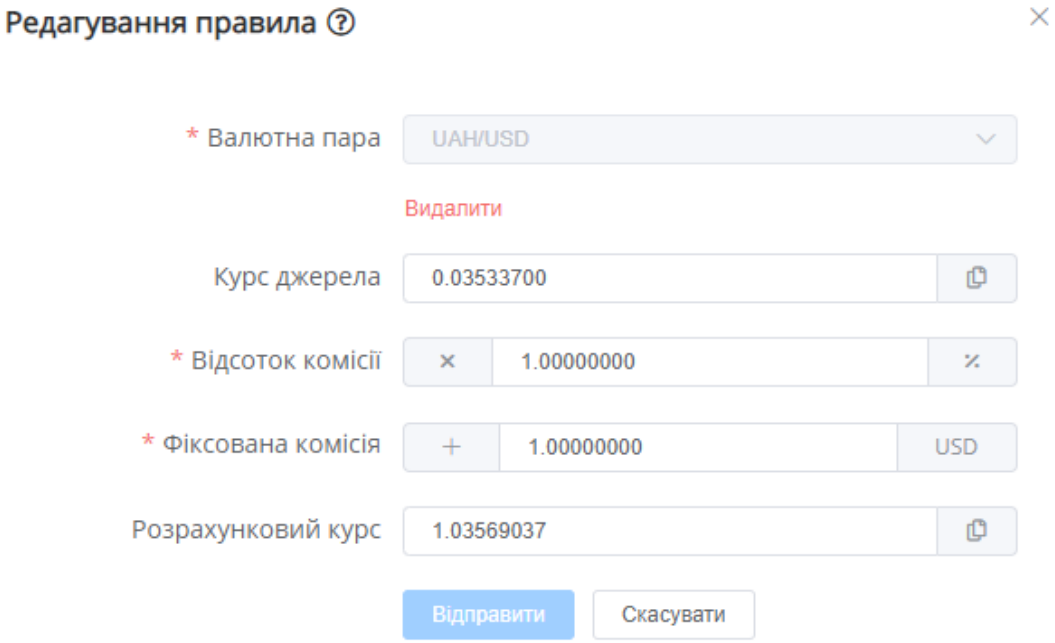

Рисунок 5.16 – Налаштування курсів валют

5.4 Робота з P2P

Одним з головних мінусів таких платежів є те, що вони не можуть існувати довго, тож необхідно власноруч контролювати це. Для цього після створення маршруту необхідно правильно його налаштувати.

На рисунку 5.17 зображено налаштування P2P опцій у платіжних маршрутах.

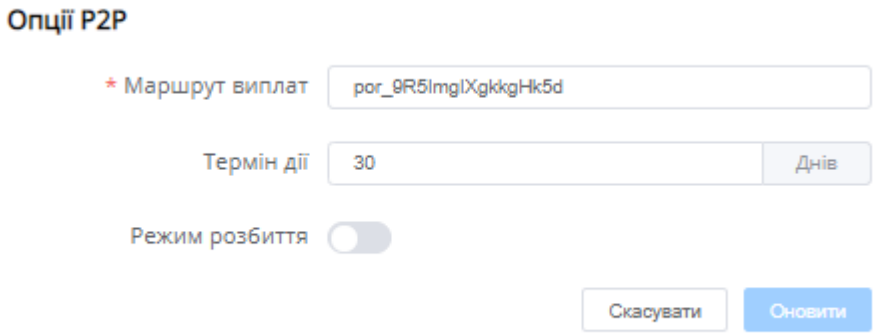

Рисунок 5.17 – Налаштування P2P опцій

Маршрут виплат необхідно вказати, щоб пов’язати виплату з платежем. Термін дії платежу необхідно дізнаватися у платіжних системах, бо від цього залежить як швидко операції будуть виплачуватися. Дуже часто гроші необхідно виплачувати на туж суму на який був платіж, але існують платіжні системи, які дозволяються розбивати виплати, для цього і існує остання опція. Розбити виплату, це коли платіж був на 500 гривень, а виплати створюються на 200, 200 та 100 гривень.

На рисунку 5.18 можна побачити, що для кожного P2P-платежу можна встановити пріоритет. Це необхідно для того, щоб у разі необхідності можна було прискорити виплату платежу.

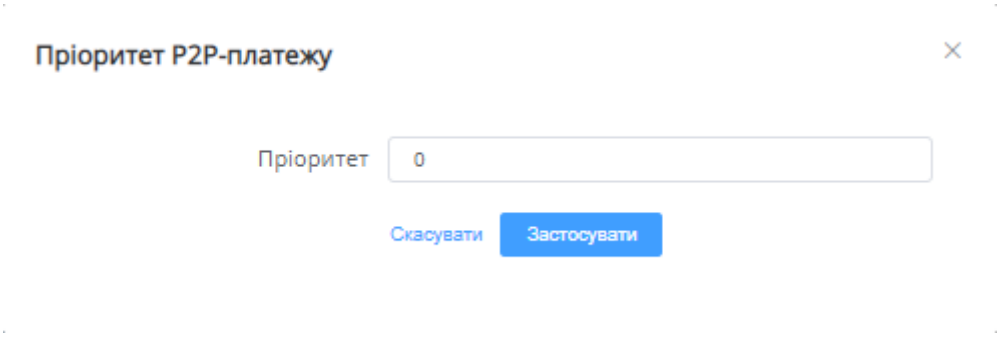

Рисунок 5.18 – Встановлення пріоритету P2P-платежу

На додатку И зображено діаграму вибору P2P-платежу. Спочатку вибираються ті, які вже половину свого строку зберігання так і не виплатились, оскільки якщо вони не виплатяться, то є вірогідність, що кошти втратяться. При умові, що такі платежі не знайдені вибираємо ті з них, які підходять нам по валюті, щоб при потребі не довелось ділити на кілька менших платежів. Якщо і такі платежі не знайдені, то це означає, що сервіс не можемо використати P2P-платежі і необхідно або виплачувати іншим способом, або фіналізувати виплату з неуспішним статусом. Коли хоч один платіж знайдено, то він проходить через деякі перевірки, такі як:

– чи задовольняє сума лімітам маршруту;
– чи необхідно використовувати розділення платежу на частини;

– чи при частковій виплаті, сума що залишилась буде задовольняти лімітам маршруту.

Якщо платіж не пройшов якусь із цих перевірок, то береться наступний і так само перевіряється, а якщо пройшов перевірку, то саме він йде на виплату.

5.5 Робота з організацією

На кожного клієнта в системі є організація, з неї відбувається основний менеджмент даних. На головному екрані виводиться та редагується основна інформація про організацію, такі як логотип, веб сайт та інша контактна інформація, він має лише інформативний характер і не має важливості для сервісу. На рисунку 5.19 можна побачити, одну з найважливіших функцій організації – зберігання так керування ключів доступу до сервісу.

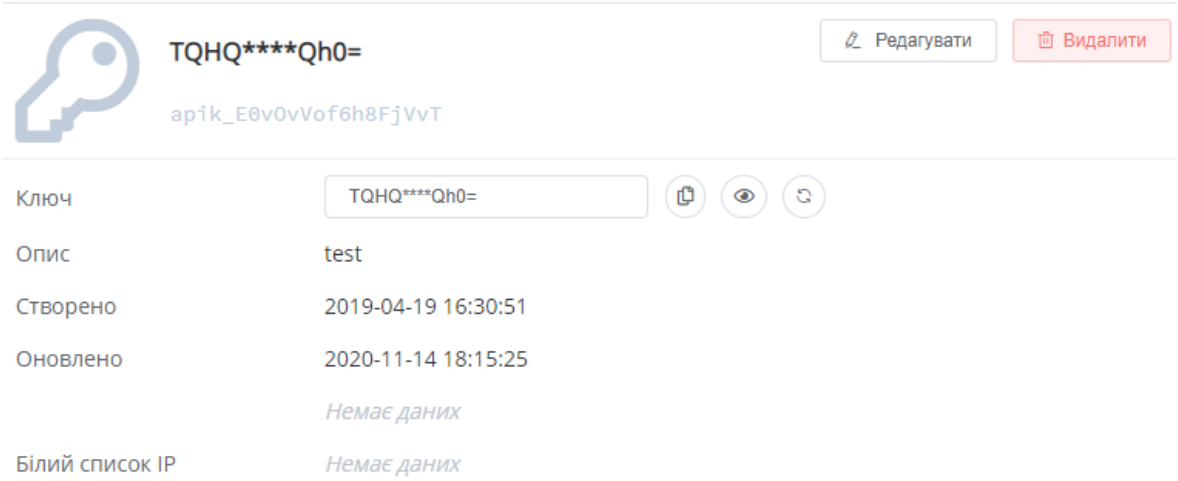

Рисунок 5.19 – Ключі доступу до організації

Ключі доступу необхідні для того, щоб робити запити по API. Оскільки в сервісі використовуються тип авторизації Basic Auth необхідно під час кожного запиту передавати його в заголовках. Якщо ключ не буде відправлено або він буде

відправлений невірно, то повернеться помилка 401, тобто що користувач не авторизований.

Під ключ також можна підв’язати список IP-адрес, з яких можна надсилати запити до сервісу. Також можна його перегенерувати, але треба буде обережним оскільки в цей момент попередній ключ одразу перестане працювати.

На рисунку 5.20 можна побачити, що ключі доступу по API мають можливості вимикати доступних до деякого функціоналу. Дана функціональність підходить для тих, хто не хоче надавати повний доступ до організації одній людині. Наприклад, якщо прийомом та виплатою займаються різні люди, то необхідно лише включити лише окремі опції, щоб у них був доступ тільки до своїх даних.

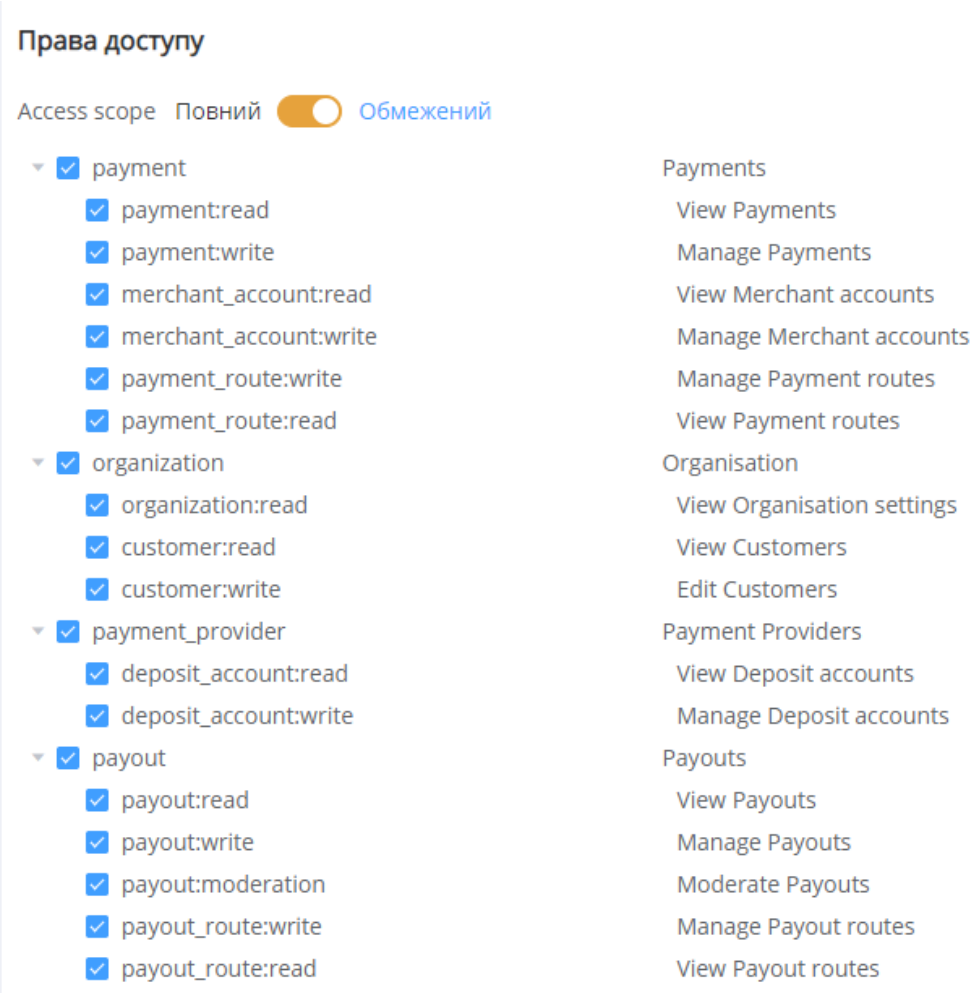

Рисунок 5.20 – Частини сервісу, до яких можна налаштувати доступи API ключів

У цьому розділі не буде детально описано принципи роботи з Basic Auth, це можна знайти в попередньому розділі, але для того, щоб було зрозуміло, як саме необхідно інтегруватися на рисунку 5.21 можна побачити, як необхідно відправляти запит з Postman[17].

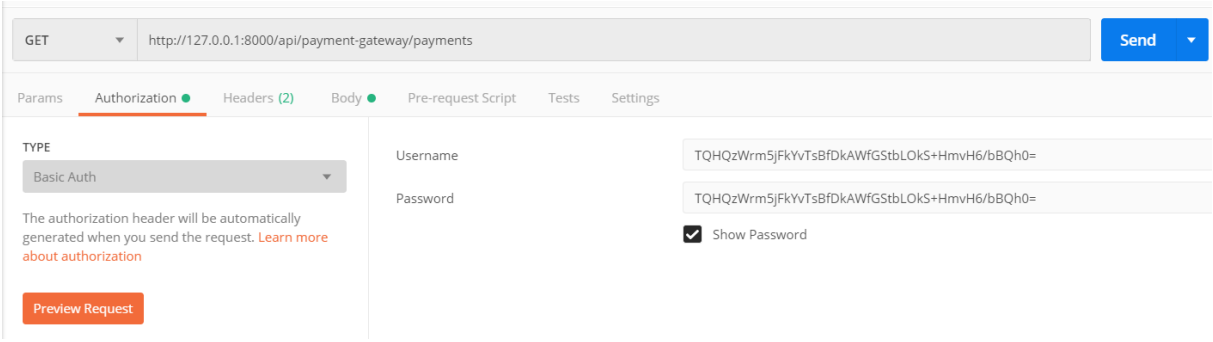

Рисунок 5.21 – Запит у застосунку Postman

Окрім роботи по API виведено окремі ролі для того, щоб взаємодіяти з маршрутизацією, платіжними аккаунтами та всім, що впливає на роботу системи. На рисунку 5.22 зображено системні ролі, які одразу доступні:

– Власник – має повний доступ до сервісу;

– Адміністратор – має повний доступ до сервісу за винятком того, що він не може змінювати власника;

– Менеджер – має всі доступи на перегляд та може модифікувати елементи сервісу, які не по пов’язані з бізнес потребами (редагувати дані організації, учасників)

– Розробник – має доступ на перегляд всього;

– Спостерігач – має доступ на перегляд за винятком конфіденційних даних, такі як ключі доступу, інтеграції тощо.

Також можна створювати свої ролі, для цього необхідно вказати, яку роль необхідно взяти за основу для створюваних. Коли вона буде створена, то необхідно зайти в неї та вибрати чи прибрати ті доступи, які вважаєте необхідним.

Це робиться за тим же принципом, як і в ключах доступу до організації як на рисунку 5.20.

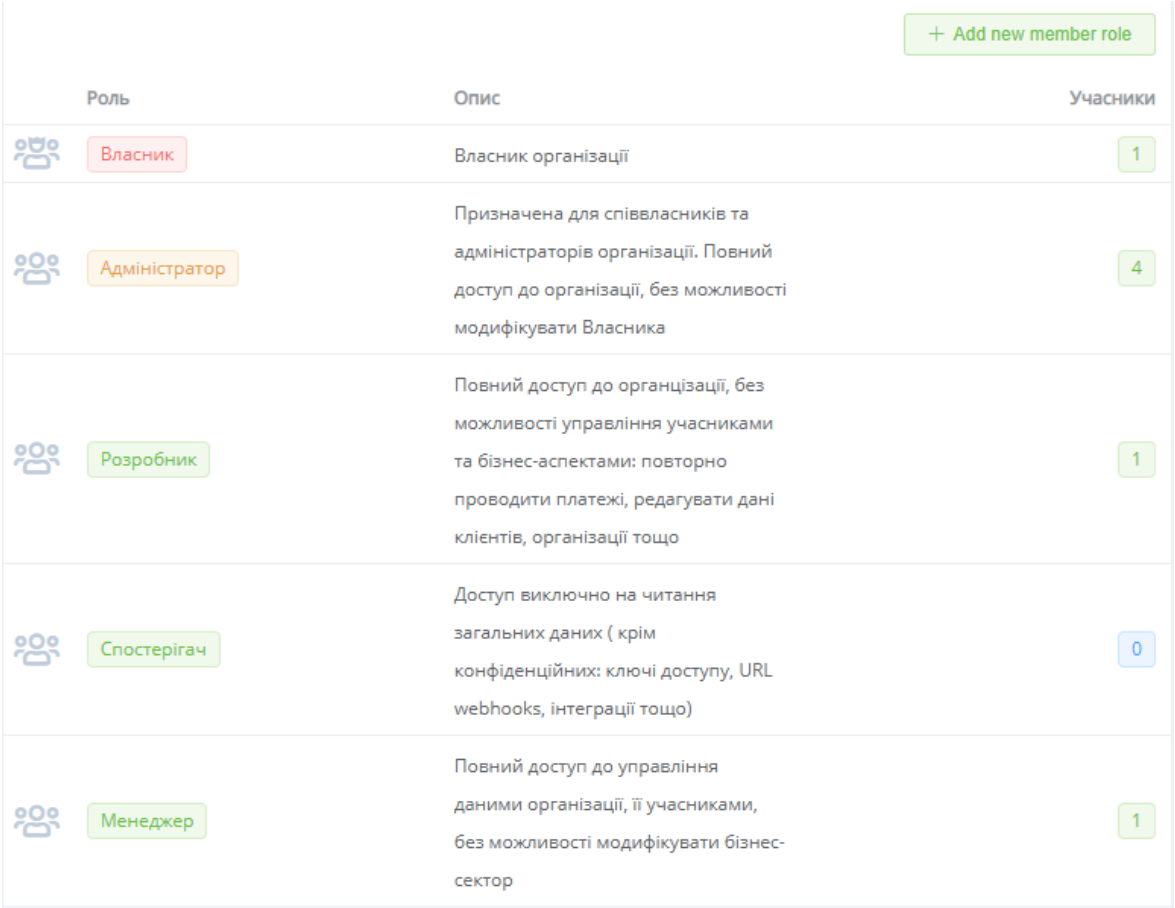

Рисунок 5.22 – Ролі за замовчуванням

Можна видаляти лише ті ролі, які були створені власноруч, але перед тим як видалити роль необхідно у всіх учасників, де вона використовується замінити роль на іншу. Якщо цього не буде зроблено то сервіс поверне помилку з ідентифікаторами клієнтів, які використовують цю роль.

Якщо є розуміння, що необхідно створювати багато різних ролей під кожного користувача, то рекомендується просто змінювати доступи самого користувача, щоб потім не загубитися серед безлічі ролей.

Висновки до розділу 5

Інструкція є закінченою, бо у неї розглянуто більшість аспектів та принципів роботи, але для того, щоб краще розуміти як пов’язані деякі речі рекомендовано також прочитати усю магістерську дисертацію, особливо розділ, де описано архітектуру. Міститься інформація про підключення та взаємодію з платіжними системами, роботу з платежами, маршрутизацією та P2P-платежами.

Все чітко поділено по пунктам, щоб була можливість дослідити кожний елемент сервісу окремо та в різні проміжки часу. Інструкції вистачить щоб повністю запустити сервіс, а вже деталі дізнатися згодом. Описано деталі взаємодії с сервісом та налаштування прав доступу до різних його частин.

6 СТАРТАП-ПРОЕКТ

В даному розділі буде описано магістерську дисертацію, як стартап-проект. Це означає, що тут описані ідеї та дії у разі виходу даного сервісу на ринок. Проведено аналіз з конкурентами, підраховано скільки необхідно витратити на перший час та визначимось у якій ніші треба себе позиціонувати та хто клієнти сервісу.

6.1 Опис ідеї проекту

В таблиці 6.1 можна побачити опис ідеї стартап-проекту.

Таблиця 6.1 – Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди користувача
	Розробити сервіс, в який можна підключати скільки завгодно платіжних систем.	Можна не підтримувати самим декілька платіжок.
	Зробити зрозумілу маршрутизацію між платіжними системами.	Не потрібно контролювати по яким маршрутам буде вигідніше.
	Робота з P2P-платежами.	Менша плата за обслуговування платежів.

тільки одну систему для платежів, а не декілька. Оскільки компаніям затратно по часу та ресурсам підключати та підтримувати декілька платіжних провайдерів то система буде затребувана.		

В таблиці 6.2 проведено аналіз сильних та слабких сторін проекту порівняно з аналогічними сервісами.

Таблиця 6.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту.

№ п/п	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Stripe	Razorpay			
1	Масштабованість	Масштабована архітектура	Масштабована система	Погано масштабована, оскільки залишається в			

				одному регіоні			
2	Маршрутизація між платіжними системами	Реалізовано	Існує можливість перенаправити користувача на інший маршрут	Не реалізовано			
3	P2P-платежі	Реалізовано взаємодія з декількома способами інтеграції	Реалізовано з декількома методами	Не реалізовано			

6.2 Технологічний аудит ідеї проекту

В таблиці 6.3 наведено приклади технологічної здійсненності проекту.

Таблиця 6.3 – Технологічна здійсненність ідеї проекту

№ п/п	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Підключення багатьох платіжних систем до сервісу	API взаємодія	Технологія наявна	Технологія доступна всім
	P2P-платежі	Платіжні системи повинні мати такий тип взаємодії	Технологія присутня приблизно у 30 % платіжних систем	Платіжні системи без перешкод надають доступ до цього функціоналу
3	Ротутинг	Технологій обходу графа	Безліч джерел в інтернеті	Купа реалізацій цієї технології в інтернеті
Обрана технологія реалізації ідеї проекту: Головними технологіями в магістерській дисертації було обрано API взаємодію, технологію обходу графу та за можливості використання P2P фунціоналу платіжних систем.				

6.3 Аналіз ринкових можливостей запуску стартап-проекту

В таблиці 6.4 проведено попередню характеристику потенційного ринку стартап-проекту.

Таблиця 6.4 – Попередня характеристика потенційного ринку стартап-проекту.

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	5
2	Загальний обсяг продаж, грн/ум.од	5 мільярдів
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Відсутні
5	Специфічні вимоги до стандартизації та сертифікації	Бажане проходження сертифікації PCI DSS
6	Середня норма рентабельності в галузі (або по ринку), %	70

В таблиці 6.5 зображено характеристика потенційних клієнтів.

Таблиця 6.5 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару

1	Правильно керувати своїми платежами.	Люди, в яких бізнес пов’язаний з онлайн платежами.	- фінансове положення споживача, можливість його регулярно оплачувати продукт.	До продукції**:** - легкий у використанні; - швидка робота; - візуальне відображення даних. До компанії-постачальника**:** - оперативна технічна підтримка; - безпека даних.
2	Використання P2P-платежів.	Бізнеси, в яких однаковий потік як платежів так і виплат.	Тип бізнесу, де це можна використовувати даний тип платежів	Однаковий потік платежів та виплат.
3	Підключення різних платіжних систем.	Бізнеси, які працюються в різних країнах або ті, які тримаються свої кошти у різних місцях.	Робота у різних країнах.	До продукції: - підтримка всіх підключених платіжних систем; - робота з різними валютами;

				До компанії-постачальника: - надавати ключі доступу до аккаунтів платіжних систем; - спілкування з представниками цих платіжних систем.

В таблиці 6.6 представлено фактори загроз стартап-проекту.

Таблиця 6.6 – Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Конкуренція	Вже є декілька готових рішень, які виконують схожі функції	Створення функцій та переваг, яких немає у конкурентів.
2	«Чутлива» інформація	Використання «чутливої» інформації (номер карт, CVV). Якщо ці дані отримає зловмисник, то довіра те репутація продукту буду під загрозою.	Забезпечення високого рівня безпеки та шифрування даних.

3	Правові чинники у роботі з даними користувача	Працювати з картами можна після проходження сертифікації PCI DSS	Консультування зі спеціалістами в сфері фінансів.
4	Знецінення ринку онлайн платежів	Якщо люди перестануть користуватися онлайн платежами то і біснесам не буде вигідно підключати такий сервіс	Популяризація онлайн платежів

В таблиці 6.7 зображено фактори можливостей.

Таблиця 6.7 – Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Збільшення онлайн платежів.	Оскільки онлайн платежів стає все більше, то це призводить до попиту у їх контролі та правильному використанню.	Пропонувати свій товар швидко розвиваючим компаніям.
2	Збільшення кібер-атак	При підвищенні кількості кібер-атак, люди переймаються за свої дані та дані їх користувачів.	Покращення безпеки системи.
3	Вихід бізнесу за рамки країни	В інтернеті рамки між країнами несуттєві, тому середні та великі біснеси можуть легко працювати в інших країнах, а це значить, що необхідно підключати різні іноземні платіжні системи	Підключення більше іноземних платіжних систем

В таблиці 6.8 проведений ступеневий аналіз конкуренції на ринку.

Таблиця 6.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Вказати тип конкуренції - досконала конкуренція	Ціна подібних систем не є фіксованою, тож кожен сам обирає ціну	Встановлення ціни на використання додатку за власним бажанням.
2. За рівнем конкурентної боротьби - міжнародний	Платіжні компанії розповсюджені о всьому світу.	Необхідність правильного таргетингу аудиторії через різні культурні та соціальні особливості.
3. За галузевою ознакою - внутрішньогалузева	Конкуренція не виходить за рамки онлайн платежів.	Аналіз ринку даної галузі з метою покращення власного продукту.
4. Конкуренція за видами товарів: - товарно-родова	Схожі системи присутні, але кожна з них займає різну тематику.	Розвиток функціоналу для збільшення кількості тематик.
5. За характером конкурентних переваг - нецінова	Застосунки модернізуються з використанням спеціально розроблених методик, поширюється	Необхідність креативного та наукового підходу до розробки можливостей застосунку.

	використання різних технологій, що покращують їх використання.	
6. За інтенсивністю - не марочна	Більш цінніше концентруватися на якості продукту, а не на рекламу.	Пріоритет на якість та безпеку розроблюваного продукту.

В таблиці 6.9 зображено аналіз конкуренції в галузі за М. Портером.

Таблиця 6.9 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	- Stripe: - Razorpay:	- лояльність клієнтів; - надійність		- потреби користувачів; - доступність цін; - якість застосунків.	- лояльність клієнтів; - привабливіша ціна
Висновки	Інтенсивність конкуренції є середньою оскільки схожих	Можливості входу на ринок існують. При розвитку		Робити нові функції безпосередньо під користувачів.	Схожі системи мають не такий високий попит, тому

	систем не так багато	застосунків вони можуть стати потенційними конкурентами на ринку та зможуть увійти на нього протягом 2-3 років.			можна без особливих труднощів увійти в цей сегмент

В таблиці 6.10 обґрунтовуються фактори, які впливають на конкурентоспроможність сервісу.

Таблиця 6.10 – Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Використання провідних технологій	Використання таких технологій як штучний інтелект, що допоможе краще використовувати можливості системи
2	Орієнтованість на споживача	Прислухання до споживачів, аналіз їх потреб та впровадження потрібних функцій у застосунок.

3	Безкоштовний перший місяць	У користувача буде можливість ознайомитися з усіма перевагами системи.

В таблиці 6.11 проводиться порівняльний аналіз сильних та слабких сторін сервісу.

Таблиця 6.11 – Порівняльний аналіз сильних та слабких сторін «Сервісу з агрегування платіжних систем»

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з агрегатором						
			–3	–2	–1	0	+1	+2	+3
1	Використання провідних технологій	15						+	
2	Орієнтованість на споживача	18							+
3	Безкоштовний перший місяць	12					+		

В таблиці 6.12 проведено SWOT- аналіз стартап-проекту.

Таблиця 6.12 – SWOT- аналіз стартап-проекту

Сильні сторони: - використання провідних технологій; - клієнт-орієнтованість; - технічна підтримка.	Слабкі сторони: - ціна на підписку; - складність системи;
Можливості:	Загрози:

- розширення функціональності; - збільшення кількості онлайн платежів; - автоматизація процесу.	- крадіжка даних - не кваліфікованість співробітників;

В таблиці 6.13 досліджено альтернативи ринкового впровадження стартап-проекту.

Таблиця 6.13 – Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Презентація на тематичних заходах для того, щоб про продукт дізналися	Дуже висока	2 місяці
2	Пропонувати продукт компаніям	Висока	6 місяців
3	Бізнес-ангел	Дуже низька	1-24 місяця

6.4 Розроблення ринкової стратегії розвитку

В таблиці 6.14 приведено вибірку цільових груп потенційних споживачів.

Таблиця 6.14 – Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової	Інтенсивність конкуренції в сегменті	Простота входу у сегмент

	потенційних клієнтів		групи (сегменту)		
1	Бізнеси з онлайн платежами	Потребують рішення в даній області - готові	Невисокий, продукт маловідомий, потрібен час	Середня, починають з'являтися альтернативи	Середня кількість конкурентів — не просто.
Які цільові групи обрано: Бізнеси з онлайн платежами					

В таблиці 6.15 визначається базова стратегія розвитку.

Таблиця 6.15 – Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1	Впровадження статистики за онлайн платежами	Акцент на надійність характеристик продукту	Аналітика над кожною платіжною системою	Стратегія диференціації
2	Використання сервісу для отримання курсів валют	Акцент на вирішення проблем обмінних установ	Курси валют беруться с різних джерел і клієнт може сам їх вибирати	Стратегія оптимальних виплат

В таблиці 6.16 визначається базова стратегія конкурентної поведінки.

Таблиця 6.16 – Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки*
1	Ні, але альтернативних рішень небагато	Буде шукати нових, більш локальних, та аналізувати конкурентів	Основний принци ведення онлайн платежів	Стратегія виклику лідеру

В таблиці 6.17 визначається стратегія позиціонування.

Таблиця 6.17 – Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
1	Висока швидкість, надійність	Стратегія диференціації	Універсальний доступ, двухфакторна автентифікація	Надійніше за конкурентів, індивідуальний підхід до клієнта, можливість удосконалення на прохання клієнта.

6.5 Розроблення маркетингової програми стартап-проекту

В таблиці 6.18 проводиться визначення ключових переваг концепції потенційного товару.

Таблиця 6.18 – Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
1	Інформативність	Аналітика системи	Велика кількість метрик, які надаються користувачам для ведення аналітики
2	Надійність	Збереження особистих даних користувачів	Файли знаходяться у зашифрованому вигляді в базі даних
3	Універсальність	Платформонезалежний доступ	Не потрібен специфічний застосунок, а тільки браузер

В таблиці 6.19 описується три рівні моделі товару.

Таблиця 6.19 – Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові
І. Товар за задумом	Платіжно-фінансова інфраструктура, яка надає єдиний інтерфейс для взаємодії з платіжними система.

II. Товар у реальному виконанні	Додаток представлений як з інтерфейсом для мобільних платформ, так і у веб—інтерфейсі. Сам інтерфейс користувача є простим, інтуїтивним, без складної логіки та недоцільних елементів. Товар є умовно безкоштовним – за розширену функціональність (наприклад, під'єднання сторонніх ресурсів) можливе за щомісячну підписку в 1000 грн.
III. Товар з підкріпленням	При купівлі підписки одразу на місяць користувач отримує знижку у 10%. Також за додаткову платню може бути налаштована платіжна система, якої немає в каталозі системи.

В таблиці 6.20 визначаються межі встановлення цін.

Таблиця 6.20 – Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1	300 — 1000 $/ місяць	10000 — 100000 $/рік	1 млн — 10 млн $/рік	500$ - 5000$/місяць

В таблиці 6.21 формується система збуту

Таблиця 6.21 – Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту

1	Орієнтація на нових клієнтів	Встановлення контактів із споживачами та підтримка їх. Формування попиту. Дослідницька робота в сфері фінансів.	0 (без посередників)…n	Самим пропонувати систему потенційним клієнтам

В таблиці 6.22 розроблюється концепція маркетингових комунікацій.

Таблиця 6.22 – Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Акцент на підтримку та вдосконалення продукту.	Формальні/ неформальні канали комунікацій	Універсальний доступ, двухфакторна автентифікація, шифрування для більшої	Інформування споживачів; Презентація на конференціях; Інформування в тематичних засобах масової інформації; Стимулювання продажу;	Даний продукт має гнучку архітектуру, що може підлаштовуватися під бажання клієнтів, має високі

			конфіденціальності	Пошук партнерів.	характеристики та надійність.

Інвестиційне забезпечення:

1) Продукт буде презентований на різних тематичних конференція(як локальних, так і міжнародних). Це дасть змогу не тільки залучити інвесторів, а і показати продукт на весь світ, що дасть змогу отримати відгук від людей, які працюють в цій сфері вже багато років.

2) Якщо не вдасться знайти інвесторів на конференціях, то можна самостійно надсилати інформацію про продукт в ті компанії, які пов’язані з онлайн платежами.

3) Спробувати знайти бізнес-ангела. Бізнес-ангел – це фізична особа, яка готова вкладати власні кошти в стартап на нульовому або початковому етапі, в обмін на частку в майбутньому підприємстві. Як показує практика, працювати з однією фізичною особою набагато легше ніж з компанію чи групою інвесторів.

Вибір договору інвестування:

1) Якщо інвестор знайдеться лише один або хтось захоче підписати ексклюзивний контракт, то буде обраний інвестиційний договір. Він дозволяє детально прописати ті цілі, на які здійснюються капіталовкладення, закріпити основні параметри і орієнтири стартаперів. У ньому можна встановити і закріпити якусь схему звітності перед інвестором, позначити кінцевий результат і терміни його досягнення, прописати обов’язки сторін тощо.

2) Якщо інвесторів буде кілька то буде укладено договір товариства. Він розроблений для того, щоб у всіх інвесторів були рівні права. Подібним способом фіксуються порядки користування майном загального підприємства, способи комунікації серед інвесторів і учасників товариства, порядок розірвання договору, виплат компенсацій тощо.

Одним з варіантів залучення інвестицій був краудфандинг.

Краудфандинг (громадське фінансування) — це співпраця людей, які добровільно об'єднують свої гроші чи інші ресурси разом, як правило через інтернет, аби підтримати зусилля інших людей або організацій.

Але такий варіант не є перспективним для нас з двох причин:

1) Тематика продукту є вузько направленою, тому вона не буде цікава звичайним людям, що за хочуть кудись вкласти гроші;

2) Нестабільність інвестувань, через відсутність договору, тобто в якийсь час можуть знизитися або взагалі пропасти інвестиції.

Робота з інвестором

Основними вимогами з нашого боку буде наступне:

1) Фінансування буде фіксоване у часі та терміні, не зважаючи на різні обставини;

2) Трафік (кількість платежів) будуть переносити на сервіс поступово. Наприклад, після завершення етапу розробки інвестор не одразу переведе всі свої платежі на нас, а лише 1%. Після тижня роботи, якщо не було виявлено помилок, то трафік підвищується до 10% и тд.

3) Інвестор не повинен впливати на шлях розвитку продукту, тобто він буде отримувати лише кошти з прибутків.

Обов’язки та права продукту

1) Продукт бере не себе усі збитки, у разі підтвердження факту їх провини.

2) Якщо у продукту не буде достатньо коштів, то може бути запропонований варіант на безкоштовну підписку протягом визначеного терміну.

3) Критична підтримка надається 24/7, всі інші питання у робочий час.

Компанії, яким би був цікавий даний стартап:

1) Letyshops – (компанія займається кешбеками) – оскільки її користувачі можуть бути з різних країн світу, то вона повинна взаємодіяти з багатьма платіжними системами.

2) Rozetka – (онлайн магазин та маркетплейс) – оскільки вона має багато продавців у різних платіжних системах, то вона може заводити кожного торговця в сервісу та бачити статистику по них, а вже на основі цієї статистики будувати аналітику.

Висновки до розділу 6

Проаналізувавши всі результати описані в даному розділі та спираючись на такі показники як динаміка ринку, попит та рентабельність можна впевнено сказати, що у сервісу є великі перспективи для його комерціалізації. Звісно, є деякі перепони для впровадження, наприклад такі як конкуренція, але завдяки конкурентоспроможності має гарні шанси на боротьбу за високі позиції в цьому сегменті.

Якщо з даним варіантом продукту щось піде не так, то можна позиціонувати себе лише тільки, як платіжний агрегатор без зайвого функціоналу значно понизивши ціну за послуги. Подальша імплементація проекту є доцільною оскільки не поступається або має переваги над конкурентами.

ВИСНОВКИ

У магістерській дисертації було розроблено сервіс з агрегування платіжних систем. У ході роботи був здійснений аналіз існуючих рішень, завдяки яким було уникнено всіх недоліків, знайдених у конкурентів сервісу. Також завдяки аналогам було виділено основні функції, які повинен виконувати сервіс. Було вирішено такі задачі, як:

- розроблення сервісу платежів;
- розроблення сервісу виплат;
- розроблення маршрутизацію між платіжними системами;
- реалізація налаштування аккаунту.

Було обрано основні засоби реалізації для стабільної роботи сервісу та обраний єдиний стандарт на всі відповіді сервісу. Приділено велику увагу обиранню СУБД, оскільки сервіс повинен витримувати великі об'єми даних не зменшуючи продуктивність.

Застосунок дозволяє підключати різні платіжні системи та додавати нові по запиту клієнтів. Створення платежів відбувається через використання платіжних сторінок, оскільки тільки такий спосіб не потребує додаткових сертифікацій. Маршрутизація зроблена так, щоб її було легко налаштовувати, щоб клієнти могли легко взаємодіяти з нею. Пояснена роботу з організацією та основними сутностями системи. З сервісом можна взаємодіяти по API не турбуючись про безпеку даних.

Розроблений додатковий, але важливий, функціонал, такий як P2P-платежі, що дозволяє економити на виплатах. Також доступна можливість виставлення комісій на певні маршрути, щоб контролювати прибуток. Написана детальна інструкція по взаємодії з сервісом та описана його архітектура.

Було детально описано та позиціонування сервісу, як стартап-проект. Провівши аналіз, можна з цілковитою впевненістю сказати, що в нього гарні шанси на зайняття свого місця на ринку онлайн платежів.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Stripe [Електронний ресурс]. – Режим доступу до ресурсу: https://stripe.com/.
2. Razorpay [Електронний ресурс]. – Режим доступу до ресурсу: https://razorpay.com/.
3. Авторський сайт IT-консультанта [Електронний ресурс] – Режим доступу до ресурсу: https://ivan-shamaev.ru/slow-php-on-windows-server-iis-fastcgi/
4. SecurityOnlone [Електронний ресурс] – Режим доступу до ресурсу: https://securityonline.info/php-7-3-is-22-faster-than-php-7-0/
5. Using symfony framework [Електронні ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Symfony.
6. Stfalcon.com [Електронний ресурс]. – Режим доступу до ресурсу: https://stfalcon.com/ru/blog/post/doctrine2-ORM-architecture.
7. Habr [Електронний ресурс]. – Режим доступу до ресурсу: https://habr.com/ru/post/282764
8. PostgresPro [Електронний ресурс]. – Режим доступу до ресурсу: https://postgrespro.ru/docs/postgrespro/9.5/textsearch-indexes
9. JsonApi [Електронний ресурс]. – Режим доступу до ресурсу: https://jsonapi.org/
10. GitHub [Електронний ресурс]. – Режим доступу до ресурсу: https://github.com/neomerx/json-api
11. Wikipedia [Електронний ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Single_sign-on
12. MDN web docs [Електронний ресурс]. – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Web/HTTP/Авторизация
13. Swagger [Електронний ресурс]. – Режим доступу до ресурсу: https://swagger.io/
14. GitHub [Електронний ресурс]. – Режим доступу до ресурсу: https://github.com/FriendsOfPHP/PHP-CS-Fixer

15. GitHub [Електронний ресурс]. – Режим доступу до ресурсу: https://github.com/phpstan/phpstan
16. Wikipedia [Електронний ресурс]. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/ISO_4217
17. Postman [Електронний ресурс]. – Режим доступу до ресурсу: https://www.postman.com/

ДОДАТОК К

Файл PayoutRoute.php

```
<?php

declare(strict_types=1);

namespace Service\PayoutBundle\Entity;

use
Component\ProcessingScheme\Model\RouteInterfac
e;
use Core\DirectoryBundle\Entity\Currency;
use
Core\DirectoryBundle\Entity\PaymentProvider;
use
Core\DirectoryBundle\Entity\PayoutMethod;
use
Core\DirectoryBundle\Entity\PayoutService;
use
Core\OrganizationBundle\Entity\Organization;
use
Core\OrganizationBundle\OrganizationRelationAwar
eInterface;
use
Core\ProviderExtensionBundle\ValueObject\Amount
Fee;
use
Core\ProviderExtensionBundle\ValueObject\Amount
Limit;
use
Core\ProviderHubBundle\Entity\ProviderAccount;
use
Core\ProviderHubBundle\Entity\DepositAccount;
use
Core\SystemBundle\Model\Timestampable;
use
Core\SystemBundle\Model\TimestampableTrait;
use
Service\PaymentBundle\Entity\PaymentRoute;
use
Component\DoctrineExtension\Generator\Uri;

/**
 * Class PayoutRoute.
 */
class PayoutRoute implements
OrganizationRelationAwareInterface,
Timestampable, RouteInterface
{
    use TimestampableTrait;

    /** @var string */
    private $id;

    /** @var ProviderAccount */
    private $account;

    /** @var Organization */
    private $organization;

    /** @var DepositAccount */
    private $depositAccount;

    /** @var string */
    private $depositAccountCurrency;

    /** @var PayoutMethod */
    private $payoutMethod;

    /** @var Currency */
    private $methodCurrency;



  /** @var PaymentProvider */

 private $paymentProvider;

    /** @var bool */
    private $enabled = true;

    /** @var bool */
    private $archived = false;

    /** @var PayoutService */
    private $service;

    /** @var bool */
    private $testMode;

    /** @var null|string */
    private $externalId;

    /** @var AmountFee */
    private $fee;

    /** @var AmountLimit */
    private $amountLimit;

    /** @var array */
    private $serviceFields = [];

    /** @var bool */
    private $p2p = false;

    /** @var null|PaymentRoute */
    private $paymentRoute;

    /**
     * PayoutRoute constructor.
     *
     * @param DepositAccount
$depositAccount
     * @param PayoutMethod   $payoutMethod
```

```
     * @param Currency      $currency
     * @param PayoutService  $service
     */
    public function __construct(
        DepositAccount $depositAccount,
        PayoutMethod $payoutMethod,
        Currency $currency,
        PayoutService $service
    ) {
        $this->id = Uri::fromPrefix('por');
        $this->initTime();
        $this->account = $depositAccount->getAccount();
        $this->paymentProvider = $this->account->getPaymentProvider();

        $this->testMode = false;
        $this->depositAccount = $depositAccount;
        $this->organization = $depositAccount->getOrganization();

        $this->depositAccountCurrency = (string) $depositAccount->getCurrency();
        $this->methodCurrency = $currency;
        $this->payoutMethod = $payoutMethod;
        $this->service = $service;

        $this->fee = new AmountFee();
        $this->amountLimit = $service->getAmountLimit();
    }

    public function getOperationAmountLimit(): AmountLimit
    {
        $amountLimit = $this->getAmountLimit();

        $payoutService = $this->getService();

        // TODO: we can add check on maximum amount by deposit account balance
        $amountMin = $amountLimit->getMin() ?? $payoutService->getAmountMin();
        $amountMax = $amountLimit->getMax() ?? $payoutService->getAmountMax();

        return new AmountLimit((float) $amountMin, (float) $amountMax);
    }

    /**
     * @return Organization
     */
    public function getOrganization(): Organization
    {
        return $this->organization;
    }

    /**
     * @return PayoutMethod
     */
    public function getPayoutMethod(): PayoutMethod
    {
        return $this->payoutMethod;
    }

    /**
     * @return string
     */
    public function getMethodCode(): string
    {
        return $this->getPayoutMethod()->getCode();
    }

    /**
     * @return PaymentProvider
     */
    public function getPaymentProvider()
    {
        return $this->paymentProvider;
    }

    /**
     * @return bool
     */
    public function isEnabled(): bool
    {
        return $this->enabled;
    }

    public function isAvailable(): bool
    {
        return $this->isEnabled();
    }

    /**
     * @param bool $enabled
     */
    public function setEnabled(bool $enabled): void
    {
        $this->enabled = $enabled;
    }

    /**
     * @return string
     */
    public function getId(): string
    {
        return $this->id;
    }

    /**
     * @return ProviderAccount
     */
    public function getAccount(): ProviderAccount
```

```
        {
            return $this->account;
        }

        /**
         * @return string
         */
        public function getDepositAccountCurrency(): string
        {
            return $this->depositAccountCurrency;
        }

        /**
         * @return Currency
         */
        public function getMethodCurrency(): Currency
        {
            return $this->methodCurrency;
        }

        /**
         * @return DepositAccount
         */
        public function getDepositAccount(): DepositAccount
        {
            return $this->depositAccount;
        }

        /**
         * @return PayoutService
         */
        public function getService(): PayoutService
        {
            return $this->service;
        }

        /**
         * @return bool
         */
        public function isTest(): bool
        {
            return $this->testMode;
        }

        /**
         * @param bool $testMode
         */
        public function setTest(bool $testMode): void
        {
            $this->testMode = $testMode;
        }

        /**
         * @return null|string
         */
        public function getExternalId(): ?string
        {
            return $this->externalId;
        }

        /**
         * @param null|string $externalId
         */
        public function setExternalId(?string $externalId): void
        {
            $this->externalId = $externalId;
        }

        /**
         * @return AmountFee
         */
        public function getFee(): AmountFee
        {
            return $this->fee;
        }

        /**
         * @param AmountFee $fee
         */
        public function setFee(AmountFee $fee): void
        {
            $this->fee = $fee;
        }

        /**
         * @return AmountLimit
         */
        public function getAmountLimit(): AmountLimit
        {
            return $this->amountLimit;
        }

        /**
         * @param AmountLimit $amountLimit
         */
        public function setAmountLimit(AmountLimit $amountLimit): void
        {
            $this->amountLimit = $amountLimit;
        }

        /**
         * @return array
         */
        public function getServiceFields(): array
        {
            return $this->serviceFields;
        }

        /**
         * @param array $serviceFields
         */
        public function setServiceFields(array $serviceFields): void
        {
```

```
        $this->serviceFields = $serviceFields;
    }

    /**
     * @return bool
     */
    public function isP2p(): bool
    {
        return $this->p2p;
    }

    /**
     * @param bool $p2p
     */
    public function setP2p(bool $p2p): void
    {
        $this->p2p = $p2p;
    }

    /**
     * @return null|PaymentRoute
     */
    public function getPaymentRoute(): ?PaymentRoute
    {
        return $this->paymentRoute;
    }

    /**
     * @param null|PaymentRoute $paymentRoute
     */
    public function setPaymentRoute(?PaymentRoute $paymentRoute): void
    {
        $this->paymentRoute = $paymentRoute;
    }

    /**
     * @return bool
     */
    public function isArchived(): bool
    {
        return $this->archived;
    }

    /**
     * @param bool $archived
     */
    public function setArchived(bool $archived): void
    {
        $this->archived = $archived;
    }
}
```

Файл PayoutRoute.orm.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<doctrine-mapping xmlns="http://doctrine-project.org/schemas/orm/doctrine-mapping"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:gedmo="http://gediminasm.org/schemas/orm/doctrine-extensions-mapping"

xsi:schemaLocation="http://doctrine-project.org/schemas/orm/doctrine-mapping
                    http://doctrine-project.org/schemas/orm/doctrine-mapping.xsd">

    <entity name="Service\PayoutBundle\Entity\PayoutRoute"
            repository-class="Service\PayoutBundle\Repository\PayoutRouteRepository"
            table="pog_payout_routes">
        <unique-constraints>
            <unique-constraint columns="method_code,organization_id,provider_code" name="ref_per_org_in_psp"/>
        </unique-constraints>

        <id name="id" type="string" length="32"/>
        <field name="enabled" type="boolean"/>
        <field name="archived" type="boolean"/>
        <field name="depositAccountCurrency" length="16"/>
        <field name="externalId" length="128"/>
        <field name="testMode" type="boolean"/>
        <field name="serviceFields" type="json_array"/>

        <embedded name="fee" column-prefix="fee_"
class="Core\ProviderExtensionBundle\ValueObject\AmountFee"/>
        <embedded name="amountLimit" column-prefix="amount_"
class="Core\ProviderExtensionBundle\ValueObject\AmountLimit"/>

        <field name="created" type="datetime_micro"/>
        <field name="updated" type="datetime">
            <gedmo:timestampable on="update"/>
        </field>
        <field name="p2p" type="boolean"/>

        <many-to-one target-entity="Core\ProviderHubBundle\Entity\DepositAccount" field="depositAccount">
            <join-column name="deposit_account_id" nullable="false"/>
        </many-to-one>
```

```
            <many-to-one target-entity="Core\DirectoryBundle\Entity\PayoutService" field="service">
                <join-column name="service_code" referenced-column-name="code" nullable="false"/>
            </many-to-one>

            <many-to-one target-entity="Core\ProviderHubBundle\Entity\ProviderAccount" field="account">
                <join-column name="provider_account_id" nullable="false"/>
            </many-to-one>

            <many-to-one target-entity="Core\OrganizationBundle\Entity\Organization" field="organization">
                <join-column name="organization_id" nullable="false"/>
            </many-to-one>

            <many-to-one target-entity="Core\DirectoryBundle\Entity\PaymentProvider" field="paymentProvider">
                <join-column name="provider_code" nullable="false" referenced-column-name="code"/>
            </many-to-one>

            <many-to-one target-entity="Core\DirectoryBundle\Entity\PayoutMethod" field="payoutMethod">
                <join-column name="method_code" nullable="false" referenced-column-name="code"/>
            </many-to-one>

            <!--<many-to-one target-entity="Core\DirectoryBundle\Entity\Currency" field="depositAccountCurrency">-->
                <!--<join-column name="account_currency_code" referenced-column-name="id" nullable="false"/>-->
            <!--</many-to-one>-->

            <many-to-one target-entity="Core\DirectoryBundle\Entity\Currency" field="methodCurrency">
                <join-column name="method_currency" referenced-column-name="code" nullable="false"/>
            </many-to-one>

            <one-to-one target-entity="Service\PaymentBundle\Entity\PaymentRoute" field="paymentRoute"/>
        </entity>

    </doctrine-mapping>
```

Файл RouteDecisionPoint.php

```
<?php

    declare(strict_types=1);

    namespace Service\PayoutRequestBundle\PayoutScheme\Service;

    use Doctrine\Common\Collections\ArrayCollection;
    use Doctrine\Common\Collections\Collection;
    use Component\ProcessingScheme\Context\DateContext;
    use Component\ProcessingScheme\Model\StrategySet;
    use Component\ProcessingScheme\RoutingScheme\Context;
    use Component\ProcessingScheme\RoutingScheme\RoutingWalker;
    use Core\ProviderExtensionBundle\Exception\AmountLimitViolationException;
    use Service\PayoutBundle\Entity\PayoutRoute;
    use Service\PayoutBundle\Repository\PayoutRouteRepository;
    use Service\PayoutRequestBundle\Entity\Enum\PayoutRequestResolution;
    use Service\PayoutRequestBundle\Entity\PayoutRequest;
    use Service\PayoutRequestBundle\Model\RouteDecision;
    use Service\PayoutRequestBundle\Model\RouteDecisionInterface;
    use Service\PayoutRequestBundle\PayoutScheme\Service\RouteFilter\RouteFilterInterface;
    use Service\PayoutRequestBundle\PayoutScheme\Service\RouteFilter\RouteFilterLogInterface;
    use Service\PayoutRequestBundle\Service\Context\PayoutRequestContext;
    use Service\PayoutRequestBundle\Service\Processor\RouteDecisionPointInterface;
    use Service\RateSchemeBundle\RateSchemeExchange;

    final class RouteDecisionPoint implements RouteDecisionPointInterface
    {
        /** @var RateSchemeExchange */
        private $exchange;
```

```
        /** @var PayoutRouteRepository */
        private $routeRepository;

        /** @var RouteFilterInterface */
        private $routeFilter;

        /** @var RoutingWalker */
        private $routingWalker;

        /** @var CardContextProvider */
        private $cardContextProvider;

        /** @var RouteStrategyProvider */
        private $strategyProvider;

        /** @var RdpLogger */
        private $logger;

        /**
         * PayoutRequestProcessor constructor.
         *
         * @param RateSchemeExchange $exchange
         * @param RoutingWalker $routingWalker
         * @param RouteFilterInterface $routeFilter
         * @param CardContextProvider $cardContextProvider
         * @param RouteStrategyProvider $strategyProvider
         * @param RdpLogger $logger
         * @param PayoutRouteRepository $routeRepository
         */
        public function __construct(
            RateSchemeExchange $exchange,
            RoutingWalker $routingWalker,
            RouteFilterInterface $routeFilter,
            CardContextProvider $cardContextProvider,
            RouteStrategyProvider $strategyProvider,
            RdpLogger $logger,
            PayoutRouteRepository $routeRepository
        ) {
            $this->exchange = $exchange;
            $this->routeFilter = $routeFilter;
            $this->routingWalker = $routingWalker;
            $this->cardContextProvider = $cardContextProvider;
            $this->strategyProvider = $strategyProvider;
            $this->routeRepository = $routeRepository;
            $this->logger = $logger;
        }

        public function getRouteDecision(PayoutRequest $payout, bool $verify = false): RouteDecisionInterface
        {
            $this->logger->initDefaultContext($payout);

            $routes = $this->routeRepository->findEnabledByService(
                $payout->getOrganization(),
                $payout->getService()->getCode(),
                $payout->isTest()
            );

            $this->logger->initialRoutes($routes);
            $strategySet = $this->provideStrategySet($payout);

            $filteredRoutes = $this->filterStrategySetRoutes($routes, $strategySet);
            $this->logger->filteredRoutes($filteredRoutes->toArray());

            $payout->initConverter($this->exchange);

            $filtrationResult = $this->routeFilter->filter($filteredRoutes, $payout);

            if ($this->routeFilter instanceof RouteFilterLogInterface) {
                $this->logger->residueRoutes($this->routeFilter->getLog());
            }

            if (!$filtrationResult->isOk()) {
                return RouteDecision::fromResolutionCode($filtrationResult->getResolution()->getValue());
            }

            $filteredRoutes = $filtrationResult->getRoutes();
            $this->logger->filteredRoutes($filteredRoutes->toArray());

            $routeDecision = $this->computeStrategySet($payout, $strategySet, $filteredRoutes);

            if (!$routeDecision->getResolution()->isOk()) {
                return $routeDecision;
            }

            /** @var PayoutRoute $route */
            $route = $routeDecision->getRoute();

            $operationAmountLimit = $route->getOperationAmountLimit();
```

```
            try {
                $blockConversion = $routeDecision->getBlockConversion() ?? $payout->calculateNextBlockConversion($route);
                $operationAmountLimit->isSatisfyWithException($blockConversion->getBlockAmount());
            } catch (\RuntimeException | AmountLimitViolationException $exception) {
                return RouteDecision::fromResolutionCode(PayoutRequestResolution::NO_ROUTES_BY_AMOUNT_LIMITS);
            }

            return $routeDecision->changeBlockConversion($blockConversion);
        }

        private function filterStrategySetRoutes(array $routesBeforeFiltration, StrategySet $strategySet): Collection
        {
            $routes = [];

            foreach ($strategySet->getStrategies() as $strategy) {
                $strategyFilter = $this->strategyProvider->get($strategy->getCode(), $strategy->getOptions());

                $filteredRoutes = $strategyFilter->filterByOptions($routesBeforeFiltration);

                $routes = array_merge($filteredRoutes, $routes);
            }

            $routes = array_unique($routes, SORT_REGULAR);

            return new ArrayCollection($routes);
        }

        private function computeStrategySet(PayoutRequest $payoutRequest, StrategySet $strategySet, Collection $routes): RouteDecision
        {
            $fallbackDecision = RouteDecision::fromResolutionCode(PayoutRequestResolution::STRATEGY_RETURNS_NO_ROUTES);

            foreach ($strategySet->getStrategies() as $strategy) {
                $this->logger->computingStrategy($strategy);

                $strategyFilter = $this->strategyProvider->get($strategy->getCode(), $strategy->getOptions());

                $decision = $strategyFilter->compute($payoutRequest, $routes);

                if ($decision->hasRoute()) {
                    break;
                }
            }

            $result = $decision ?? $fallbackDecision;
            $this->logger->receivingDecision($result);

            return $result;
        }

        private function provideStrategySet(PayoutRequest $payoutRequest): StrategySet
        {
            $routingScheme = $payoutRequest->getServiceScheme()->getRoutingScheme();
            $this->initContext($payoutRequest);

            $strategySet = $this->routingWalker->walk($routingScheme);
            $this->logger->routingResult($strategySet);

            return $strategySet;
        }

        private function initContext(PayoutRequest $payoutRequest): void
        {
            $providers = [
                new DateContext(),
                new PayoutRequestContext($payoutRequest, $this->exchange),
            ];

            $cardContext = $this->cardContextProvider->extractFromPayoutRequest($payoutRequest);

            null === $cardContext ?: $providers[] = $cardContext;

            $context = new Context($providers);

            $this->routingWalker->appendContext($context);
        }
    }
```

Файл Organization.php

```
<?php

declare(strict_types=1);

namespace Core\OrganizationBundle\Entity;

use
Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use
Core\CallbackBundle\Enum\WebhookVersionEnum;
use
Core\OrganizationBundle\Dto\OrganizationEncrypti
onKey;
use
Core\OrganizationBundle\Dto\OrganizationEncrypti
onKeys;
use
Core\OrganizationBundle\Dto\OrganizationFeatures;
use
Core\SystemBundle\ImageAwareInterface;
use Core\UserBundle\Entity\User;
use
Service\PaymentRequestBundle\Service\SecurityKey
Generator;
use
Paymaxi\Bundle\AccountBundle\Model\Account;
use
Paymaxi\Bundle\ApiBundle\Model\ApiUserInterface
;
use
Paymaxi\Component\DoctrineExtension\Generator\
Uri;

/**
 * Class Organization.
 */
class Organization extends Account
{
    /** @var string */
    private $email;

    /** @var string */
    private $phone;

    /** @var string */
    private $supportUrl;

    /** @var string */
    private $site;

    /** @var string */
    private $contactEmail;

    /** @var string */
    private $address;

    /** @var string */
    private          $webhookVersion          =
WebhookVersionEnum::V2;

    /** @var bool */
    private $hasLogo = false;

    /** @var ArrayCollection|MemberRole[] */
    private $roles;

    /** @var string */
    private $testSecret;

    /** @var string */
    private $liveSecret;

    const KEY_PREFIX = 'ok';

    /** @var array|null */
    private $features;

    /** @var null|string */
    private $domainName;

    /** @var null|string */
    private $pspDirectoryCode;

    /** @var array */
    private $encryptionKeys = [];

    /**
     *                                          @see
\Paymaxi\Bundle\AccountBundle\Factory\AccountF
actory
     * Organization constructor.
     */
    public function __construct()
    {
        parent::__construct();
        $this->created = new \DateTime();
        $this->id = Uri::fromPrefix('org');
        $this->roles = new ArrayCollection();
        $this->refreshSecrets();
    }

    /**
     * @return ApiUserInterface[]|Collection
     */
    public function getApiKeys(): Collection
    {
        return $this->apiKeys;
    }

    /**
     * @param OrganizationApiKey|string $key
     *
     * @return bool
     */
    public function hasApiKey($key): bool
    {
        if ($key instanceof OrganizationApiKey)
{
            $key = $key->getId();
        }
```

```
            return $this->apiKeys->exists(function ($i, OrganizationApiKey $apiKey) use ($key) {
                return $apiKey->getId() === $key;
            });
        }

        /**
         * @param string $key
         *
         * @return null|OrganizationApiKey
         */
        public function getApiKey(string $key): ?OrganizationApiKey
        {
            $apiKey = $this->apiKeys
                ->filter(function (OrganizationApiKey $apiKey) use ($key) {
                    return $apiKey->getId() === $key;
                })->first();

            return false === $apiKey ? null : $apiKey;
        }

        /**
         * @return User
         */
        public function getOwner(): User
        {
            /** @var OrganizationMember $member */
            $member = $this->getMembers()->filter(function (OrganizationMember $member) {
                return $member->getMemberRole()->isOwner();
            })->first();

            return $member->getUserEntity();
        }

        /**
         * @return Collection
         */
        public function getUsers()
        {
            $collection = $this->getMembers()->map(function (OrganizationMember $member) {
                return $member->getUser();
            });

            return $collection;
        }

        /**
         * @return bool
         */
        public function hasLogo(): bool
        {
            return $this->hasLogo;
        }

        /**
         * @param bool $status
         */
        public function setLogo(bool $status): void
        {
            $this->hasLogo = $status;
        }

        /**
         * @return string
         */
        public function getEmail(): ?string
        {
            return $this->email;
        }

        /**
         * @return string
         */
        public function getPhone(): ?string
        {
            return $this->phone;
        }

        /**
         * @return string
         */
        public function getSupportUrl(): ?string
        {
            return $this->supportUrl;
        }

        /**
         * @param string $email
         */
        public function setEmail(?string $email): void
        {
            $this->email = $email;
        }

        /**
         * @param string $phone
         */
        public function setPhone(?string $phone): void
        {
            $this->phone = $phone;
        }

        /**
         * @param string $supportUrl
         */
        public function setSupportUrl(?string $supportUrl): void
        {
            $this->supportUrl = $supportUrl;
        }

        /**
         * @param User $user
```

```
     *
     * @return bool
     */
    public function isCreator(User $user): bool
    {
        /** @var User $creator */
        $creator = $this->creator;

        return $creator->getId() === $user->getId();
    }

    /**
     * @param string $userId
     *
     * @return null|OrganizationMember
     */
    public function getMember(string $userId): ?OrganizationMember
    {
        /** @var OrganizationMember $member */
        foreach ($this->getMembers()->toArray() as $member) {
            /** @var null|User $user */
            $user = $member->getUser();

            if (null === $user) {
                continue;
            }

            if ($user->getId() === $userId) {
                return $member;
            }
        }

        return null;
    }

    public function hasMember(OrganizationMember $organizationMember): bool
    {
        return $this->getMembers()->exists(function ($index, OrganizationMember $organizationMemberStorage) use ($organizationMember) {
            return $organizationMemberStorage->getId() === $organizationMember->getId();
        });
    }

    /**
     * @return null|string
     */
    public function getContactEmail(): ?string
    {
        return $this->contactEmail;
    }

    /**
     * @param null|string $contactEmail
     */
    public function setContactEmail(?string $contactEmail): void
    {
        $this->contactEmail = $contactEmail;
    }

    /**
     * @return null|string
     */
    public function getSite(): ?string
    {
        return $this->site;
    }

    /**
     * @param null|string $site
     */
    public function setSite(?string $site): void
    {
        $this->site = $site;
    }

    /**
     * @return null|string
     */
    public function getAddress(): ?string
    {
        return $this->address;
    }

    /**
     * @param null|string $address
     */
    public function setAddress(?string $address): void
    {
        $this->address = $address;
    }

    /**
     * @param string $version
     */
    public function setWebhookVersion(string $version): void
    {
        $this->webhookVersion = $version;
        if (WebhookVersionEnum::V2 === $version && $this->secretsAreEmpty()) {
            $this->refreshSecrets();
        }
    }

    /**
     * @return string
     */
    public function getWebhookVersion(): string
    {
        return $this->webhookVersion;
```

```
    }

    /**
     *                                   @return
ArrayCollection|Collection|MemberRole[]
     */
    public function getRoles(): Collection
    {
        return $this->roles;
    }

    /**
     * @param MemberRole $memberRole
     *
     * @return Organization
     */
    public function addRole(MemberRole
$memberRole)
    {
        if (!$this->hasRole($memberRole)) {
            $this->roles->add($memberRole);
        }

        return $this;
    }

    /**
     * @param MemberRole $memberRole
     *
     * @return bool
     */
    public function hasRole(MemberRole
$memberRole): bool
    {
        return $this->roles->exists(function
($index, MemberRole $memberRoleStorage) use
($memberRole) {
            return $memberRoleStorage->getId()
=== $memberRole->getId();
        });
    }

    /**
     * @return null|string
     */
    public function getTestSecret(): ?string
    {
        return $this->testSecret;
    }

    /**
     * @return null|string
     */
    public function getLiveSecret(): ?string
    {
        return $this->liveSecret;
    }

    /**
     * @return bool
     */
    protected function secretsAreEmpty(): bool
    {
        return null === $this->testSecret &&
null === $this->liveSecret;
    }

    protected function refreshSecrets(): void
    {
        $this->testSecret =
$this->generateKey('test');
        $this->liveSecret =
$this->generateKey('live');
    }

    /**
     * @param string $name
     *
     * @return string
     */
    protected function generateKey(string
$name): string
    {
        return \implode('_', [self::KEY_PREFIX,
$name, SecurityKeyGenerator::generate(32)]);
    }

    /**
     * @return OrganizationFeatures
     */
    public function getFeatures():
OrganizationFeatures
    {
        return null === $this->features ? new
OrganizationFeatures() :
OrganizationFeatures::createFromArray($this-
>features);
    }

    /**
     * @param OrganizationFeatures $features
     */
    public function
setFeatures(OrganizationFeatures $features): void
    {
        $this->features = $features->toArray();
    }

    /**
     * @return null|string
     */
    public function getDomainName(): ?string
    {
        return $this->domainName;
    }

    /**
     * @param null|string $domainName
     */
    public function setDomainName(?string
$domainName): void
    {
```

```
        $this->domainName = $domainName;
    }

    /**
     * @return null|string
     */
    public function getPspDirectoryCode(): ?string
    {
        return $this->pspDirectoryCode;
    }

    /**
     * @return OrganizationEncryptionKeys
     */
    public function getEncryptionKeys(): OrganizationEncryptionKeys
    {
        return OrganizationEncryptionKeys::fromArray($this->encryptionKeys);
    }

    /**
     * @param int $revision
     *
     * @return OrganizationEncryptionKey
     */
    public function getEncryptionKeyByRevision(int $revision): OrganizationEncryptionKey
    {
        $keys = OrganizationEncryptionKeys::fromArray($this->encryptionKeys);
        if ($revision === $keys->getActive()->getRevision()) {
            return $keys->getActive();
        }

        if (null !== $keys->getPrevious() && $revision === $keys->getPrevious()->getRevision()) {
            return $keys->getPrevious();
        }
        throw new \RuntimeException('Wrong organization key revision');
    }

    /**
     * @param OrganizationEncryptionKey $encryptionKey
     */
    public function setEncryptionKey(OrganizationEncryptionKey $encryptionKey): void
    {
        $keys = OrganizationEncryptionKeys::fromArray($this->encryptionKeys);
        $keys->setNewKey($encryptionKey);
        $this->encryptionKeys = $keys->toArray();
    }
}
```

Файл WeightRouteStrategy.php

```
<?php

declare(strict_types=1);

namespace Service\PayoutRequestBundle\RoutingScheme\RouteStrategy;

use Doctrine\Common\Collections\Collection;
use Component\RoutingScheme\Model\RouteInterface;
use Component\RoutingScheme\RouteStrategy\ConstraintAwareInterface;
use Component\RoutingScheme\RouteStrategy\OptionsAwareInterface;
use Service\PayoutRequestBundle\Entity\Enum\PayoutRequestResolution;
use Service\PayoutRequestBundle\RoutingScheme\Model\RouteDecision;
use Symfony\Component\Validator\Constraints\All;
use Symfony\Component\Validator\Constraints\Collection as ConstraintCollection;
use Symfony\Component\Validator\Constraints\Count;
use Symfony\Component\Validator\Constraints\Range;
use Symfony\Component\Validator\Constraints\Type;
use Webmozart\Assert\Assert;

/**
 * Class WeightRouteStrategy.
 */
final class WeightRouteStrategy implements PayoutRouteStrategyInterface, OptionsAwareInterface, ConstraintAwareInterface
{
    private const ROUTE = 'route';
    private  const WEIGHT = 'weight';

    /** @var array */
    private $options = [];

    /**
     * @param Collection $routes
     *
     * @return RouteDecision
     */
```

```
        public function compute(Collection $routes): RouteDecision
        {
            if ($routes->isEmpty()) {
                throw new \InvalidArgumentException('Collection must have at least one element.');
            }

            if (1 === $routes->count()) {
                return new RouteDecision(new PayoutRequestResolution(PayoutRequestResolution::OK), $routes->first());
            }

            $options = \array_filter($this->options, function ($option) use ($routes) {
                $id = $option[self::ROUTE];

                return false !== $routes->filter(function (RouteInterface $payoutRoute) use ($id) {
                    return $payoutRoute->getId() === $id;
                })->first();
            });

            if (0 === \count($options)) {
                throw new \InvalidArgumentException('Options can not be empty.');
            }

            $routeId = null;

            $multiplier = 10000000;
            $total = \array_sum(\array_column($options, self::WEIGHT));
            $r = \random_int(1, $total * $multiplier);

            $acc = 0;

            foreach ($options as $option) {
                $acc += $option[self::WEIGHT] * $multiplier;
                if ($acc >= $r) {
                    $routeId = $option[self::ROUTE];
                    break;
                }
            }

            $delegateTo = new DirectRouteStrategy();
            $delegateTo->setOptions([DirectRouteStrategy::ROUTE => $routeId]);

            return $delegateTo->compute($routes);
        }

        /**
         * @return array
         */
        public function getOptions(): array
        {
            return $this->options;
        }

        /**
         * @param array $options
         */
        public function setOptions(array $options): void
        {
            Assert::allKeyExists($options, self::ROUTE);
            Assert::allKeyExists($options, self::WEIGHT);

            $this->options = $options;
        }

        public function getConstraints($routeConstraint)
        {
            return [
                new Type('array'),
                new Count(['max' => 20]),
                new All([
                    new ConstraintCollection([
                        self::ROUTE => $routeConstraint,
                        self::WEIGHT => [new Type('integer'), new Range(['min' => 1, 'max' => 1000])],
                    ]),
                ]),
            ];
        }
    }
```